

Tissue P Systems with Symport/Antiport Rules of One Symbol Are Computationally complete

Rudolf Freund^{a,*} Marion Oswald^a

^a*Faculty of Informatics, Vienna University of Technology, Favoritenstr. 9–11,
A-1040 Vienna, Austria*

Abstract

We consider tissue P systems using symport/ antiport rules of only one symbol where in each link (channel) between two cells at most one rule is applied, but in each channel a symport/ antiport rule has to be used if possible. We prove that any recursively enumerable set of k -dimensional vectors of natural numbers can be generated (accepted) by such a tissue P system with symport/ antiport rules of one symbol using at most $2k + 5$ (at most $3k + 7$) cells.

Key words: Antiport Rules, Membrane computing, P systems, Turing computability

1 Introduction

In the area of membrane computing there are two main classes of systems: P systems with a hierarchical (tree-like) structure as already introduced in the original paper of Gheorghe Păun (see [8]) and tissue P systems with cells arranged in an arbitrary graph structure (see [5]). We here consider tissue P systems using symport/antiport rules for the communication between cells (these communication rules first were investigated in [7]).

It is well known that equipped with the maximally parallel derivation mode (tissue) P systems with only one membrane (one cell) already reach universal

* Corresponding author

Email addresses: rudi@emcc.at (Rudolf Freund), marion@emcc.at (Marion Oswald).

computational power, even with antiport rules of weight two (e.g., see [2] and [3]); yet on the other hand, in these P systems the number of symbols remains unbounded (at least if we fix the initial multisets).

Considering the generation of recursively enumerable sets of natural numbers we may also ask the question how many symbols we need for obtaining computational completeness. In [10] the quite surprising result was proved that three symbols are enough in the case of P systems with symport/ antiport rules. The specific type of maximally parallel application of at most one rule in each connection (link) between two cells or one cell and the environment, respectively, in tissue P systems allows for an even more surprising result: The minimal number of one symbol is already sufficient to obtain computational completeness, e.g., we will show that any recursively enumerable set of natural numbers can be generated by a tissue P system with at most seven cells using symport/ antiport rules of only one symbol.

2 Preliminaries

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [1] and [12]. We just list a few notions and notations: N denotes the set of positive integers; the set of non-negative integers (natural numbers) is denoted by N_0 . V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ , as identity; by RE ($RE(k)$) we denote the family of recursively enumerable languages (over a k -letter alphabet); for the corresponding families of λ -free languages we write RE^+ and $RE^+(k)$, respectively. By $\Psi_T(L)$ we denote the Parikh image of the language $L \subseteq T^*$, and by $PsFL$ we denote the set of Parikh images of languages from a given family FL . $PsRE(k)$ ($PsRE^+(k)$) corresponds with the family of recursively enumerable sets of k -dimensional vectors of non-negative (positive) integers.

2.1 Register machines

The proofs of the main results established in this paper are based on the simulation of register machines; we refer to [6] for original definitions, and to [2] for definitions like those we use in this paper:

A (*non-deterministic*) register machine is a construct $M = (n, R, l_0, l_h)$, where n is the number of registers, R is a finite set of instructions injectively labelled with elements from a given set $lab(M)$, l_0 is the initial/start label, and l_h is the final label.

The instructions are of the following forms:

- $l_1 : (A(r), l_2, l_3)$,
Add 1 to the contents of register r and proceed to one of the instructions (labelled with) l_2 and l_3 . (We say that we have an ADD instruction.)
- $l_1 : (S(r), l_2, l_3)$,
If register r is not empty, then subtract 1 from its contents and go to instruction l_2 , otherwise proceed to instruction l_3 . (We say that we have a SUB instruction.)
- $l_h : halt$,
Stop the machine. The final label l_h is only assigned to this instruction.

A register machine M is said to generate a vector (s_1, \dots, s_k) of natural numbers if, starting with the instruction with label l_0 and all registers containing the number 0, the machine stops (it reaches the instruction $l_h : halt$) with the first k registers containing the numbers s_1, \dots, s_k (and all other registers being empty).

Without loss of generality, in the succeeding proofs we will assume that in each ADD instruction $l_1 : (A(r), l_2, l_3)$ and in each SUB instruction $l_1 : (S(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct (for a short proof see [4]).

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate exactly the sets of vectors of natural numbers which can be generated by Turing machines, i.e., the family *PsRE*. Especially we know that three registers are enough to generate any recursively enumerable set of positive integers, where, moreover, the only instructions on the first register are ADD instructions (see [6], [2]).

2.2 Tissue P systems with symport/ antiport rules

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [9]; comprehensive information can be found on the P systems web page <http://psystems.disco.unimib.it>.

Tissue P systems were introduced in [5], and tissue-like P systems with channel states were investigated in [4]. Here we deal with the following type of systems omitting the channel states:

A *tissue P system* (of degree $m \geq 1$) with symport/ antiport rules is a construct

$$\Pi = \left(O, T, E, w_1, \dots, w_m, ch, (R(i, j))_{(i, j) \in ch} \right),$$

where O is the alphabet of *objects*, $T \subseteq O$ is the alphabet of *terminal* objects, $E \subseteq O$ is the set of objects present in arbitrarily many copies in the environment, w_1, \dots, w_m are strings over O representing the *initial* multiset of *objects* present in the cells of the system (it is assumed that we have m cells, labelled with $1, 2, \dots, m$), $ch \subseteq \{(i, j) \mid i, j \in \{0, 1, 2, \dots, m\}, (i, j) \neq (0, 0)\}$ is the set of links (*channels*) between cells (these were called *synapses* in [4]; 0 indicates the environment), $R(i, j)$ is a finite set of antiport rules of the form x/y , for some $x, y \in O^*$, associated with the channel $(i, j) \in ch$.

An *antiport rule* of the form $x/y \in R(i, j)$ for the ordered pair (i, j) of cells means moving the objects specified by x from cell i (from the environment, if $i = 0$) to cell j , at the same time moving the objects specified by y in the opposite direction. The rules with one of x, y being empty are, in fact, *symport rules*, but we do not always explicitly consider this distinction here, as it is not relevant for what follows. For short, we shall also speak of *tissue P system* only when dealing with *tissue P system with symport/ antiport rules* as defined above.

The objects from E are never exhausted, irrespective of how many copies of each of them are brought into the system, arbitrarily many copies remain available in the environment.

The computation starts with the multisets specified by w_1, \dots, w_m in the m cells; in each time unit, a rule is used on each channel for which a rule can be used (if no rule is applicable for a channel, then no object passes over it). Therefore, the use of rules is sequential at the level of each channel, but it is parallel at the level of the system: all channels which can use a rule must do it (the system is synchronously evolving). The computation is successful if and only if it halts.

Usually the result of a halting computation is the vector which describes the multiplicity of objects from T present in a designated output cell i_o in the halting configuration (the objects from $O - T$ are ignored when considering the result). As in this paper we are dealing with systems using only one symbol, we consider the contents of the first k cells to represent the k components of a k -dimensional vector of natural numbers. The set of all k -dimensional vectors computed in this way by the system Π is denoted by $Ps(\Pi, k)$. The family of sets $Ps(\Pi, k)$ of vectors computed as above by systems with at most m cells is denoted by $PsOtP(k, m)$. When any of the parameters k, m is not bounded, it is replaced by $*$.

In [4], only channels (i, j) with $i \neq j$ are allowed, and, moreover, for any i, j only one channel out of $\{(i, j), (j, i)\}$ is allowed, i.e., between two cells (or one

cell and the environment) only one channel is allowed (as we shall see in the next section, this technical detail may influence the number of cells needed to obtain computational completeness). The family of sets $Ps(\Pi, k)$ of vectors computed as above by such tissue P systems with at most m cells is denoted by $PsOtP(k, m)$.

Convention. As we are dealing with tissue P systems with only one object b , we shall only write

$$\Pi = \left(w_1, \dots, w_m, ch, (R(i, j))_{(i, j) \in ch} \right)$$

as well as x/y for the antiport rule b^x/b^y and x for any multiset b^x .

3 Results

We first prove our main result that one symbol is enough for obtaining computational completeness in such a general form that not only the generative case, but also the accepting case and the case of computing functions can be covered by this proof:

Theorem 1 $PsRE(k) = PsOtP(k, m)$ for all $k \geq 1$ and $m \geq 2k + 8$.

PROOF. We only prove the inclusion $PsRE(k) \subseteq PsOtP(k, 2k+8)$. To this aim, let us consider a register machine $M = (n, R, l_0, l_h)$ generating the set of k -dimensional vectors $N(M)$, for some $k \geq 1$, for which we now construct the tissue P system (of degree $2k + 8$)

$$\Pi = \left(w_1, \dots, w_{2k+8}, ch, (R(i, j))_{(i, j) \in ch} \right)$$

as follows:

The cells 1 to k represent the output registers and start with the initial value 0, whereas the cells $k + 1$ and $k + 2$ represent the two additional registers and start with the initial value 1 (because we need a bias of one to be able to simulate SUB instructions). The cells $k + 2 + i$ with $1 \leq i \leq k$ are used for simulating the ADD instructions on the registers i ; the cells $2k + 3$ and $2k + 4$ are needed for simulating incrementing and decrementing registers $k + 1$ and $k + 2$, respectively, whereas the cells $2k + 5$ and $2k + 6$ are used for testing these registers for zero (which for the corresponding cells means that the contents is 1). The cell $2k + 7$ is the *program cell* which guides the simulation of the program of the register machine; the last cell $2k + 8$ is a *trap* which, whenever started, leads to a non-halting computation by using the antiport rule $2/2$

in the channel $R(2k+8, 0)$. The initial values of the cells are listed in the following table (the value $c(l_0)$ will be explained later):

$$\begin{aligned}
w_i &= 0 \text{ for } 1 \leq i \leq k, \\
w_i &= 1 \text{ for } k+1 \leq i \leq k+2, \\
w_i &= 0 \text{ for } k+3 \leq i \leq 2k+6, \\
w_{2k+7} &= 2c(l_0), \\
w_{2k+8} &= 1.
\end{aligned}$$

According to the description of the functioning of the cells given above, we define the following channel structure:

$$\begin{aligned}
ch &= \{(2k+7, j) \mid j \in \{0, 2k+8\} \cup \{i \mid k+3 \leq i \leq 2k+6\}\} \\
&\cup \{(k+2+j, j), (k+2+j, 2k+8) \mid 1 \leq j \leq k+2\} \\
&\cup \{(k+4+j, j), (k+4+j, 2k+8) \mid k+1 \leq j \leq k+2\} \\
&\cup \{(2k+8, 0)\}.
\end{aligned}$$

The most important part of the proof is to define a suitable encoding c for the instructions of the register machine: Without loss of generality we assume the labels of M to be positive integers such that the labels assigned to ADD and SUB instructions have the values $7i-6$ for $1 \leq i < t$, as well as $l_0 = 1$ and $l_h = 7t-6$, for some $t \geq 1$.

We now define the encoding c on positive integers in such a way that the function $c : N \rightarrow N$ is strictly monotonic and increasing sufficiently to guarantee that only taking the right channel rules leads to a computation not activating the trap; we shall argue later why the chosen function fulfills the necessary conditions, and we emphasize correct argumentation instead of searching for smaller function values; thus, to be on the safe side, also for the subsequent proofs, we take the following function c :

$$c(0) = 99, \quad c(i+1) = 2 \sum_{j=0}^i (c(j) + c(0)) \text{ for } i \geq 0.$$

With $l_0 = 1$ we therefore obtain $w_{2k+7} = 8c(0) = 792$.

Equipped with this coding function we are now able to define the following sets of rules in the corresponding channels for simulating the actions of the given register machine M :

- (1) Throughout the whole computation in Π , the program cell will contain a value $2c(l)$ for some $l \leq 7t-6$; in order to guarantee the correct sequence

of encoded rules the trap is activated in case of a wrong choice, which in any case leaves some symbols in the program cell to activate the trap by the rule

$$1/0 \text{ in } R(2k + 7, 2k + 8).$$

Moreover, the trap can also be activated by the rules

$$1/0 \text{ in } R(k + 2 + r, 2k + 8), 1 \leq r \leq k + 2, \text{ and}$$

$$2/1 \text{ in } R(2k + 4 + s, 2k + 8), 1 \leq s \leq 2.$$

- (2) As already mentioned above, the trap contains the initial value 1 and is activated by any additional symbol arriving there due to the rule

$$2/2 \text{ in } R(2k + 8, 0),$$

which guarantees that after the activation of the trap the computation can never halt.

- (3) For each ADD instruction $l_1 : (A(r), l_2, l_3)$ of M , we introduce the rules

$$c(l_1)/0 \text{ and}$$

$$0/(c(l_1) - 1) \text{ in } R(2k + 7, k + 2 + r)$$

as well as the rule

$$1/0 \text{ in } R(k + 2 + r, r).$$

Moreover, the program cell $2k + 7$ controls the simulation via the rules

$$c(l_1)/(2c(l_1 + 1)),$$

$$2c(l_1 + 1)/(2c(l_2) - (c(l_1) - 1)) \text{ and}$$

$$2c(l_1 + 1)/(2c(l_3) - (c(l_1) - 1)) \text{ in } R(2k + 7, 0).$$

In that way, the ADD instruction $l_1 : (A(r), l_2, l_3)$ of M is simulated in two steps: In the first step, half of the contents of the program cell is used to get the next code $2c(l_1 + 1)$, whereas the second half is sent to cell $k + 2 + r$ there “activating” the rule $1/0 \text{ in } R(k + 2 + r, r)$, which in the second step increments the contents of cell r (which represents register r); the remaining rest $c(l_1) - 1$ in the second step goes back from cell $k + 2 + r$ to the program cell - by the rule $0/(c(l_1) - 1) \text{ in } R(2k + 7, k + 2 + r)$ - and together with $2c(l_x) - (c(l_1) - 1)$ brought into the program cell in the second step - by the rule $2c(l_1 + 1)/(2c(l_x) - (c(l_1) - 1)) \text{ in } R(2k + 7, 0)$ for $x \in \{2, 3\}$ - there forms the starting value $2c(l_x)$ for the simulation of the next instruction labelled by l_x .

If in any of these steps described above we do not choose the correct rule, then the trap will be activated: The coding function has been chosen in such a way that - especially in the program cell, but even more obviously in cell $k + 2 + r$ - instead of the correct rule for the labels l_1 and $l_1 + 1$ only rules for labels $l < l_1$ or $l < l_1 + 1$, respectively, could be chosen, but on the other hand, even when using all possible links, all these numbers $c(l)$ in sum could not exhaust the total number of symbols currently present in the program cell or cell $k + 2 + r$, hence, symbols would remain and activate the trap.

Obviously, in the second step the single symbol not going back from cell $k + 2 + r$ to the program cell could also activate the trap, but then the computation would enter an infinite loop at this point.

- (4) For simulating the decrementing step of a SUB instruction $l_1 : (S(r), l_2, l_3)$

from R - observe that this only means $r \in \{k + 1, k + 2\}$ - we introduce the rules

$$\begin{aligned} & c(l_1) / 0, \\ & c(l_1 + 1) / (c(l_1) - 1) \text{ and} \\ & 0 / (c(l_1 + 1) + 2) \text{ in } R(2k + 7, k + 2 + r) \end{aligned}$$

as well as the rule

$$1/2 \text{ in } R(k + 2 + r, r)$$

for decrementing the contents of cell r (which represents register r).

Moreover, the program cell $2k + 7$ controls the simulation via the rules

$$\begin{aligned} & c(l_1) / (2c(l_1 + 1)), \\ & c(l_1 + 1) / (2c(l_1 + 2) - (c(l_1) - 1)), \text{ and} \\ & 2c(l_1 + 2) / (2c(l_2) - (c(l_1 + 1) + 2)) \text{ in } R(2k + 7, 0). \end{aligned}$$

In that way, the decrementing step of the SUB instruction $l_1 : (S(r), l_2, l_3)$ of M now is simulated in three steps: In the first step, half of the contents of the program cell is used to get the next code $2c(l_1 + 1)$, whereas the second half is sent to cell $k + 2 + r$ there “activating” the rule $1/2$ in $R(k + 2 + r, r)$, which in the second step decrements the contents of cell r (which represents register r); the remaining rest $c(l_1) - 1$ in the second step is exchanged with $c(l_1 + 1)$ by the rule $c(l_1 + 1) / (c(l_1) - 1)$.

Now the rule $0 / (c(l_1 + 1) + 2)$ from $R(2k + 7, k + 2 + r)$ is only applicable if the decrementing rule $1/2$ from $R(k + 2 + r, r)$ has been applied in the second step - e.g., applying the incrementing rule $1/0$ instead would lead to a situation where the number in cell $k + 2 + r$ is a little bit too small, but too large to be consumed otherwise, hence, this would lead to an activation of the trap in the third step. Now $c(l_1 + 1) + 2$ goes back from cell $k + 2 + r$ to the program cell - by the rule $0 / (c(l_1 + 1) + 2)$ in $R(2k + 7, k + 2 + r)$ - and together with $2c(l_2) - (c(l_1 + 1) + 2)$ brought into the program cell in the third step - by the rule $2c(l_1 + 2) / (2c(l_2) - (c(l_1 + 1) + 2))$ in $R(2k + 7, 0)$ - there forms the starting value $2c(l_2)$ for the next instruction labelled by l_2 .

Again we notice that if in any of these steps described above we do not choose the correct rule, then the trap is activated.

- (5) For simulating the zero test, i.e., the case where the contents of register r is zero, of a SUB instruction $l_1 : (S(r), l_2, l_3)$ from R we now need the additional intermediate cells for the registers $k + 1$ and $k + 2$, i.e., the cells $2k + 5$ and $2k + 6$, respectively. The simulation of the zero test case takes four steps with the following rules:

$$\begin{aligned} & c(l_1) / 0, \\ & 0 / (c(l_1) - 1) \\ & c(l_1 + 2) / 0 \text{ and} \\ & 0 / (c(l_1 + 2) + 1) \end{aligned}$$

are taken into $R(2k + 7, k + 4 + r)$, the rule

$$1/2 \text{ into } R(k + 4 + r, r),$$

and the program cell $2k + 7$ controls the simulation via the rules

$$c(l_1) / (2c(l_1 + 1)),$$

$$\begin{aligned}
& c(l_1 + 1) / (2c(l_1 + 2) - (c(l_1) - 1)), \\
& c(l_1 + 2) / (2c(l_1 + 3)), \text{ and} \\
& (2c(l_1 + 3)) / (2c(l_3) - (c(l_1 + 2) + 1)) \text{ in } R(2k + 7, 0).
\end{aligned}$$

In that way, the zero test step of the SUB instruction $l_1 : (S(r), l_2, l_3)$ of M now is simulated in four steps: In the first step, half of the contents of the program cell is used to get the next code $2c(l_1 + 1)$, whereas the second half is sent to cell $k + 4 + r$ there putting a checking symbol for possibly “activating” the rule $1/2$ in $R(k + 4 + r, r)$, which in the second step would decrement the contents of cell r (which represents register r); the remaining rest $c(l_1) - 1$ in the second step is sent back to the program cell where it sums up to $2c(l_1 + 2)$ together with $2c(l_1 + 2) - (c(l_1) - 1)$ brought in there by the rule $c(l_1 + 1) / (2c(l_1 + 2) - (c(l_1) - 1))$.

In the third step, $2c(l_1 + 2)$ splits again with one half going to cell $k + 4 + r$ and the second half bringing in $2c(l_1 + 3)$ from the environment by the rule $c(l_1 + 2) / (2c(l_1 + 3))$. In the meantime, the trap has not been activated only if the application of the rule $1/2$ in $R(k + 4 + r, r)$ has not been possible in the second step in cell $k + 4 + r$.

Now $c(l_1 + 2) + 1$ goes back from cell $k + 4 + r$ to the program cell - by the rule $0 / (c(l_1 + 2) + 1)$ in $R(2k + 7, k + 4 + r)$ - and together with $2c(l_3) - (c(l_1 + 2) + 1)$ brought into the program cell in the third step - by the rule $2c(l_1 + 3) / (2c(l_3) - (c(l_1 + 2) + 1))$ in $R(2k + 7, 0)$ - there forms the starting value $2c(l_3)$ for the next instruction labelled by l_3 .

Once again we notice that if in any of these steps described above we do not choose the correct rule, then the trap is activated.

- (6) Finally, for the halt label $l_h = 7t - 6$ we only take the rule

$$(2c(l_h)) / 0 \text{ into } R(2k + 7, 0),$$

hence, the work of Π will stop exactly when the work of M stops (provided the trap has not been activated due to a wrong non-deterministic choice during the computation).

The explanations given above show that $N(M) = Ps(\Pi, k)$. □

Whereas our first theorem already shows that the number of cells can be bounded, on the other hand either the weight of the antiport rules to be used cannot be bounded as this follows from the preceding proof or else the weight of the initial multisets inside the cells at the beginning of a computation in the tissue P system cannot be bounded (e.g., in case we simulate the actions of a register machine simulating a universal Turing machine and put the code of the Turing machine to be simulated into register $k + 1$).

In the one-dimensional case, some even more sophisticated ideas allow us to reduce the number of cells compared with the number of cells as constructed in the proof of Theorem 1 considerably from 10 to 7 :

Theorem 2 $PsRE(1) = PsOtP(1, m)$ for all $m \geq 7$, i.e., for any recursively enumerable set L of non-negative integers we can construct a tissue P system with at most 7 cells that generates L .

PROOF. We again start with a register machine $M = (n, R, l_0, l_h)$ generating the set L of one-dimensional vectors, and as we know, without loss of generality, we may assume that $n \leq 3$; hence, from the proof of Theorem 1, we immediately infer that the tissue P system has at most 10 cells.

Yet there are some possibilities for considerable improvements that allow us to decrease the number of cells from 10 to 7:

Cell 1 represents the output register, the cells 2 and 3 represent the additional two registers, cell 4 and 5 are assigned to cells 2 and 3 for zero testing as in the proof of Theorem 1, cell 6 is the program cell, and cell 7 is the trap.

The first observation is that the only instructions working on the first register are ADD instructions, which can be performed directly from the program cell, i.e., using the rules 1/0 in channels (6, 1) and (6, 7) as well as $(c(l_1) - 1) / c(l_2)$ and $(c(l_1) - 1) / c(l_3)$ in channel (6, 0) for every ADD instruction $l_1 : (A(1), l_2, l_3)$.

But also incrementing and decrementing of cells 2 and 3 can be accomplished directly from the program cell if we use a suitable linear encoding $\alpha x + \beta$ of the contents x of the corresponding register; for register 2 (3) its contents x is represented in cell 2 (3) as $4x + 2$ ($12x + 6$).

Hence, we now construct the tissue P system (of degree 7)

$$\Pi = \left(0, 2, 6, 1, 5, 2c(l_0), 1, ch, (R(i, j))_{(i, j) \in ch} \right)$$

as follows:

$$\begin{aligned} ch &= \{(6, j) \mid j \in \{0, 1, 2, 3, 4, 5, 7\}\} \\ &\cup \{(4, 2), (5, 3), (4, 7), (5, 7), (7, 0)\}. \end{aligned}$$

With the following sets of rules in the corresponding channels the actions of the given register machine M can be simulated:

$$\begin{aligned}
R(4, 2) &= \{2/6\}, \\
R(4, 7) &= \{3/0\}, \\
R(5, 3) &= \{6/18\}, \\
R(5, 7) &= \{7/0\}, \\
R(6, 1) &= \{1/0\}, \\
R(6, 2) &= \{4/0, 2/6\}, \\
R(6, 3) &= \{12/0, 6/18\}, \\
R(6, 7) &= \{1/0\}, \\
R(7, 0) &= \{2/2\},
\end{aligned}$$

and the “program” rules

$$\begin{aligned}
R(6, 0) &= \{(2c(l_1) - 1) / (2c(l_2)), (2c(l_1) - 1) / (2c(l_3)) \mid \\
&\quad l_1 : (A(1), l_2, l_3) \in R\} \\
&\cup \{(2c(l_1) - 4) / (2c(l_2)), (2c(l_1) - 4) / (2c(l_3)) \mid \\
&\quad l_1 : (A(2), l_2, l_3) \in R\} \\
&\cup \{(2c(l_1) - 12) / (2c(l_2)), (2c(l_1) - 12) / (2c(l_3)) \mid \\
&\quad l_1 : (A(3), l_2, l_3) \in R\} \\
&\cup \{(2c(l_1) - 2) / (2c(l_1 + 1)), (2c(l_1 + 1) + 6) / (2c(l_2)) \mid \\
&\quad l_1 : (S(2), l_2, l_3) \in R\} \\
&\cup \{(2c(l_1) - 6) / (2c(l_1 + 1)), (2c(l_1 + 1) + 18) / (2c(l_2)) \mid \\
&\quad l_1 : (S(3), l_2, l_3) \in R\} \\
&\cup \{c(l_1) / (2c(l_1 + 1)), c(l_1 + 1) / (2c(l_1 + 2) - (c(l_1) - 1)), \\
&\quad c(l_1 + 2) / (2c(l_3) - (c(l_1 + 2) + 1)) \mid \\
&\quad l_1 : (S(r), l_2, l_3) \in R, r \in \{2, 3\}\} \\
R(6, 4) &= \{c(l_1) / 0, c(l_1 + 1) / (c(l_1) - 1), (c(l_1 + 1) + 1) / 0 \mid \\
&\quad l_1 : (S(r), l_2, l_3) \in R, r \in \{2, 3\}\}, \\
R(6, 5) &= \{c(l_1) / 0, c(l_1 + 1) / (c(l_1) - 1), (c(l_1 + 1) + 1) / 0 \mid \\
&\quad l_1 : (S(r), l_2, l_3) \in R, r \in \{2, 3\}\}.
\end{aligned}$$

The main observation for proving the correctness of the construction is that, whenever several symbols are left in the program cell for simulating incre-

menting or decrementing of a register, these symbols cannot be split to initiate other actions, because in that case the total number of symbols would not be exhausted thus activating the trap, too; e.g., 12 symbols to be used for the simulation of incrementing register three using $12/0$ from $R(6, 3)$ could be “misused” for applying $1/0$ from $R(6, 1)$, $4/0$ from $R(6, 2)$, and $6/18$ from $R(6, 3)$, which in sum only consumes 11 symbols so that $1/0$ from $R(6, 7)$ could be applied, too, thus activating the trap 7.

The argumentation concerning the encoding function c runs as in the proof of Theorem 1. The simulation of incrementing or decrementing a register r now takes only one step directly involving the corresponding cell (representing this register r) from the program cell, whereas zero checking (of register 2 or register 3) still runs over an additional cell, but now takes only three steps: In the first two steps, one additional symbol is brought to cell 4 or cell 5, respectively, thus possibly activating the rules $2/6$ and $6/18$, respectively. In the third step it is checked that the corresponding decrementing rules has not been applied by reducing the number of remaining symbols to the initial value again; in case that the decrementing rule had been applicable, now the trap would be activated, too.

In sum, the actions of the given register machine M can be simulated correctly; in case of a wrong non-deterministic choice for the rules to be applied in a maximally parallel way in the tissue P system Π , the trap is activated, thus prohibiting wrong derivations to halt. Hence, we infer $N(M) = Ps(\Pi, k)$. These observations conclude the proof. \square

We can take over the tricky details for the first output register as well as for the two last registers for the general case of $PsRE(k)$, too, hence, from Theorem 1 we immediately obtain the following result:

Corollary 3 $PsRE(k) = PsOtP(k, m)$ for all $k \geq 1$ and $m \geq 2k + 5$.

Omitting the condition that for any i, j only one channel out of $\{(i, j), (j, i)\}$ is allowed, we can at least save one cell, i.e., the one used as the trap:

Corollary 4 $PsRE(k) = PsO'tP(k, m)$ for all $k \geq 1$ and $m \geq 2k + 4$.

PROOF. In the proofs of the preceding theorems and corollaries we can omit the cell acting as a trap t and instead use the idea of blowing up the contents of every cell s not representing a register by using a connection $(0, s)$ in case that superfluous symbols occur in the cell: Instead of a rule x/y in $R(s, t)$ we now use the rules $(2c(l_h + 2))/x$ in $R(0, s)$ and $(2c(l_h + 2))/c(l_h + 2)$ in $R(0, t)$, which guarantees that in this cell the rule $(2c(l_h + 2))/c(l_h + 2)$ can be applied forever as soon as this cell has been “activated as a trap” by the

application of the rule $(2c(l_h + 2))/x$ in $R(0, s)$ - which corresponds to the activation of the trap by applying the rule x/y in $R(s, t)$ in the original proof.

□

4 Further Variants

Obvious variants are obtained by considering the accepting mode and the computing mode: In the accepting case, we designate the first l cells as the input cells, and start the computation by introducing multisets over a singleton alphabet in these cells; these multisets are accepted if and only if the computation halts. The main difficulty is that now all registers to be simulated have to allow for decrementing which either needs a lot more cells as indicated in the proof of Theorem 1 or else we have to allow a linear encoding of the input values as used in the proof of Theorem 2. Obviously, similar problems arise in the computing case.

4.1 Accepting tissue P systems with one symbol

In the accepting case, all registers have to allow for decrementing and zero tests. Therefore, according to the construction given in the proof of Theorem 1, we need $3(k + 2) + 2$ cells for accepting sets from $PsRE^+(k)$ (if we allow the bias of one symbol, then this holds true for sets from $PsRE(k)$, too).

Theorem 5 *For every set L from $PsRE^+(k)$ we can construct a tissue P system with at most $3k + 7$ cells using symport/ antiport rules of only one symbol which accepts L .*

PROOF. As all registers have to allow for decrementing and zero tests, we can use the construction for the registers $k + 1$ and $k + 2$ as given in the proof of Theorem 1 for all $k + 2$ registers, which in sum together with the program cell and the trap yields the necessity to use at least $3(k + 2) + 2$ cells. Yet (at least) one cell can be saved by omitting the cell used for incrementing and decrementing register one, which instead can be done directly via the program cell as elaborated in the proof of Theorem 2. □

We should like to mention that again we can save (at least) one cell when allowing two channels between two cells or one cell and the environment.

4.2 Computing tissue P systems with one symbol

For computing functions from N^α to N^β we can use the first α cells as input registers and the first β cells as output registers; due to preceding constructions we obtain the following result:

Theorem 6 *For every function f from N^α to N^β we can construct a tissue P system with at most $3k + 7 - \max\{0, \beta - \alpha\}$ cells using symport/ antiport rules of only one symbol which computes f .*

PROOF. According to the arguments given in the proof of Theorem 5, all registers having to allow for decrementing and zero tests need 3 cells for their simulation, whereas in the case $\beta - \alpha > 0$ we need one cell less for each of the $\beta - \alpha$ registers only acting as output registers. As argued in the proof of Theorem 5, for one register we may again directly organize incrementing and decrementing in the program cell. Hence, in sum together with the program cell and the trap we need $3k + 7 - \max\{0, \beta - \alpha\}$ cells. \square

Again we can save (at least) one cell when allowing two channels between two cells or one cell and the environment

5 Open Questions

The number of cells needed in the proofs given in this paper possibly may be improved. Moreover, we may consider variants where no symport rules and only pure antiport rules are used; in this case, every cell allowing for being involved in antiport rules initially has to contain at least one symbol, i.e., we can only generate sets from $PsRE^+$.

We may also consider tissue P systems with specific structures, e.g., with a tree structure, and pose the question whether computational completeness can also be obtained under the constraint of a given graph pattern for the cell structure. In the case of a tree structure this corresponds to a model of P systems with rules assigned to membranes (e.g. see [2]) where in each derivation step at most one rule has to be applied at each membrane, but in a maximally parallel manner with respect to the involved membranes.

As was proved in [10], three symbols are enough to obtain computational completeness in the case of P systems with the rules acting in a maximally parallel manner; in that case, the minimal number of symbols needed obviously

seems to be two, and it is an interesting open question whether two symbols are enough or not to obtain computational completeness.

References

- [1] J. Dassow, Gh. Păun, Regulated Rewriting in Formal Language Theory, Springer-Verlag, Berlin, 1989.
- [2] R. Freund, M. Oswald, P Systems with activated/prohibited membrane channels, in [11], pp. 261–268.
- [3] R. Freund, A. Păun, Membrane systems with symport/ antiport rules: universality results, in [11], pp. 270–287.
- [4] R. Freund, Gh. Păun, M.J. Pérez-Jiménez, Tissue-like P systems with channel states, Brainstorming Week on Membrane Computing, Sevilla, February 2004, TR 01/04 of Research Group on Natural Computing, Sevilla University (2004) 206–223 and Theoretical Computer Science, in press.
- [5] C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Paton, Tissue P systems, Theoretical Computer Science 296 (2) (2003) 295–326.
- [6] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
- [7] A. Păun, Gh. Păun, The power of communication: P systems with symport/ antiport, New Generation Computing, 20, 3 (2002) 295–306.
- [8] Gh. Păun, Computing with membranes, Journal of Computer and System Sciences 61, 1 (2000) 108–143 and TUCS Research Report 208 (1998) (<http://www.tucs.fi>)
- [9] Gh. Păun, Computing with Membranes: An Introduction, Springer-Verlag, Berlin, 2002.
- [10] Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodríguez-Patón, Symport/ antiport P systems with three objects are universal, downloadable from [13].
- [11] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing. International Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers, Lecture Notes in Computer Science, Vol. 2597, Springer-Verlag, Berlin, 2003.
- [12] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages (3 volumes), Springer-Verlag, Berlin, 1997.
- [13] The P Systems Web Page, <http://psystems.disco.unimib.it>