# On Communication in Tissue P Systems: Conditional Uniport

Sergey Verlan[1], Francesco Bernardini[2], Marian Gheorghe[3], Maurice Margenstern[4]

[1]LACL, Département Informatique, Université Paris 12
61 av. Général de Gaulle, 94010 Crétiel, France
E-mail: verlan@univ-paris12.fr

[2]Leiden Institute of Advanced Computer Science, Universiteit Leiden
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
E-mail: bernardi@liacs.nl

[3]Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
E-mail: M.Gheorghe@dcs.shef.ac.uk

[4]Université Paul Verlaine - Metz, UFR MIM, LITA, EA 3097
Ile du Saulcy, 57045 Metz Cédex, France
E-mail: margens@univ-metz.fr

**Abstract.** The paper introduces (purely communicative) tissue P systems with conditional uniport. Conditional uniport means that every application of a communication rule moves one object in a certain direction by possibly using another one as an activator which is left untouched in the place where it is. Tissue P systems with conditional uniport are shown to be computationally complete in the sense that they can recognise all recursively enumerable sets of natural numbers. This is achieved by simulating deterministic register machines.

## 1 Introduction

Membrane computing is an emerging branch of natural computing which deals with distributed and parallel computing devices of a bio-inspired type, which are called *membrane systems* or *P systems* (see [14], [15], and also [1] for a comprehensive bibliography of P systems). P systems, originally devised by Gh. Păun in [14], are introduced as computing devices which abstract from the structure and functioning of living cells - they are defined as a hierarchical arrangement of regions delimited by membranes (*membrane structure*), with each region having associated a multiset of objects and a finite set of rules.

Communication of objects through membranes is one of the fundamental features of every membrane system and this naturally led to the question of closely investigating the power of communication, that is, considering *purely communicative P systems* where objects are never modified but they just change their place within the system. A first attempt to address this issue was done in [11] by

considering certain "vehicle-objects" (an abstraction for plasmids or for vectors from gene cloning) which actively carry objects through membranes. Then, the bio-chemical idea of membrane transport of pairs of molecules was considered in [13] by introducing the notion of *P systems with symport/antiport*. When two chemicals can pass through a membrane only together, in the same direction, the process is called *symport*; when the two chemicals can pass only with the help of each other, but in opposite directions, we say that we have an *antiport* process (see [2]). Such a mechanism has been formalised and generalised in P systems by considering multisets of objects, here denoted by $x, y$, that are moved across the membranes by means of rules of the form $(x, in)$, $(x, out)$ (*symport rules*), and $(x, out; y, in)$ (*antiport rules*); $x, y$ can be of any size but they cannot be empty. As happens in few other models (e.g., see the billiard ball model [19], the chip firing game [9], or railway simulation [10]), computing by communication turns out to be computationally complete: by using only symport and antiport rules, we can generate all Turing computable sets of numbers [13]. However, as in the aforementioned models of other inspiration, in order to generate all Turing computable sets, some infinity must be present and, in P systems with symport/antiport, this is provided in the form of an infinite supply of objects taken from an external environment. Several subsequent works have been dedicated to improve this result in what concerns both the number of membranes used and the size of symport/antiport rules used inside the membranes. We refer to [17] for a survey of these investigations.

The class of membrane computing models was later extended to *tissue P systems* [15]: a variant of P systems where the underlying structure is defined as an arbitrary graph. Nodes in the graph represent *cells* (i.e., elementary membranes) that are able to communicate objects alongside the edges of this graph [15]. From a biological point of view, tissue P systems are seen as an abstract model of cells in multicellular organisms where they form a multitude of different tissues, arranged into organs performing various functions [2]. In particular, purely communicative tissue P systems with symport/antiport were investigated in [15] by showing completeness results for systems using rules of different sizes and different structures for the underlying graph. More recently, it was proved in [3] that tissue P systems with symport/antiport rules of a minimal size (i.e., rules of the forms $(a, in)$, $(a, out)$, $(a, out; b, in)$, with $a, b$ objects from a given alphabet) are computational complete and two cells suffice.

In this paper we consider tissue P systems with a different model of communication called *conditional uniport*. Conditional uniport means that every application of a communication rule moves one object in a certain direction by possibly using another one as an activator which is left untouched in the place where it is. In other words, rules are assigned to the edges of the graph and they represent channels of communication; in the cell placed at one end of an edge, an object is used to activate the channel and either receive another object (in a single copy) from the cell at the other end of that edge, or send another object (in one single copy) to the cell at the other end of the edge. The biological motivation for conditional uniport is twofold. On the one hand, in living cells, the active trans-

port of small molecules is driven by proteic channels: a molecule binds to one of these channels which, through a change of conformation, is able to release the molecule outside the compartment (see [2]). On the other hand, in tissues, cell-to-cell communication depends heavily on extracellular signal molecules, which are produced by a cell to signal their neighbours or cells that are further away. In turn, these cells can respond to extracellular signal molecules by means of particular proteins called receptors; each receptor binds at cell-surface level to particular signal molecules and it is able to initiate a specific response inside the cell (see [2]). In both cases, it is only the small molecule or the signal molecule to be moved in one direction and, in order to be transported or to be recognised, it requires another object, a proteic channel or a receptor. Such a model of communication was already investigated in [4] where an evolution-communication model was considered. This means that, besides conditional uniport, multiset rewriting rules can be used to modify the objects placed inside the cells. The main result reported in [4] then showed how to achieve computational completeness by having only 2 cells and using non-cooperative multiset rewriting rules. Here we instead focus on the purely communicative case, with the usual assumption of an infinite supply of objects from the environment, and we show that tissue P systems with conditional uniport are able to simulate deterministic register machines by using 24 cells. This means that they can recognise all recursively enumerable sets of natural numbers.

## 2 Preliminaries

We recall here some basic notions concerning the notation commonly used in membrane computing and the few notions of formal language theory we need in the rest of the paper. We refer to [15], [18] for further details.

An alphabet is a finite non-empty set of abstract symbols. Given an alphabet $V$, we denote by $V^*$ the set of all possible strings over $V$, including the empty string $\lambda$. The length of a string $x \in V^*$ is denoted by $|x|$ and, for each $a \in V$, $|x|_a$ denotes the number of occurrences of the symbol $a$ in $x$. A multiset over $V$ is a mapping $M : V \longrightarrow \mathbb{N}$ such that, $M(a)$ defines the multiplicity of $a$ in the multiset $M$ ($\mathbb{N}$ denotes the set of natural numbers). Such a multiset can be represented by a string $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$ and by all its permutations, with $a_j \in V$, $M(a_j) \neq 0$, $1 \leq j \leq n$. In other words, each string $x \in V^*$ identifies a multiset over $V$ defined by $M_x = \{ (a, |x|_a) \mid a \in V \}$.

We also recall the notion of a (deterministic) *register machine* [12]. A deterministic *register machine* is the following construction:

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

where $Q$ is a set of states, $R = \{A_1, \dots, A_k\}$ 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. $(p, A+, q) \in P$, $p, q \in Q, p \neq q, A \in R$ (being in state $p$, increase register $A$ and go to state $q$).

2. $(p, A-, q, s) \in P$, $p, q, s \in Q, A \in R$ (being in state $p$, decrease register $A$ and go to $q$ if successful or to $s$ if $A$ is zero).
3. $(q_f, STOP)$.

Moreover, for each state $p \in Q$, there is only one instruction of one of the above types.

A configuration of a register machine is given by the $k+1$-tuple $(q, n_1, \ldots, n_k)$ 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 $M$ computes a value $y \in \mathbb{N}$ on the *input* $x \in \mathbb{N}$ if starting from the initial configuration $(q_0, x, 0, \ldots, 0)$ it reaches the final configuration $(q_f, y, 0, \ldots, 0)$. We say that $M$ recognises the set $S \subseteq \mathbb{N}$ if for any input $x \in S$ the machine stops and for any $y \notin S$ the machine performs an infinite computation. It is known that register machines recognise all recursively enumerable sets of numbers [12].

We may also consider non-deterministic register machines where the first type of instruction is of the form $(p, A+, q, s)$ and with the following meaning: if the machine is in state $p$, then the register $A$ is increased and the current state is changed to $q$ or $s$ non-deterministically. In this case the result of the computation is the set of all values of the first register when, starting with the configuration $(q_0, 0, 0, \ldots, 0)$, the computation eventually halts. We assume that the machine empties all registers except the first register before stopping. It is known that non-deterministic register machines generate all recursively enumerable sets of non-negative natural numbers starting from empty registers [12].

## 3 The Model

Now we formally introduce the notion of tissue P systems with conditional uniport.

**Definition 1.** *A* tissue P systems with conditional uniport *(a* TPCU*, for short) of degree $n \geq 1$ is a construct*

$$\mathcal{T} = (V, E, w_1, \ldots, w_n, R, c_I)$$

*where:*

1. *$V$ is a* finite alphabet*;*
2. *$E \subseteq V$ is the* set of symbols which appear in the environment*;*
3. *$w_i \in V^*$, for all $1 \leq i \leq n$, is the multiset* initially associated to cell *$i$;*
4. *$R$ is a finite set of* rules *of the forms:*
   *(a) $(i, a \to j)$ with $1 \leq i, j \leq n$, $i \neq j$ and $a \in V$,*
   *(b) $(i, a \to j, b)$ with $1 \leq i, j \leq n$, $i \neq j$ and $a, b \in V$,*
   *(c) $(i, b, a \to j)$ with $1 \leq i, j \leq n$, $i \neq j$ and $a, b \in V$,*
   *(d) $(0, a \to j, b)$ with $1 \leq j \leq n$ and $a, b \in V$,*
   *(e) $(i, b, a \to 0)$ with $1 \leq i \leq n$ and $a, b \in V$,*

*(f)* $(i, a \to 0)$ *with* $1 \leq i \leq n$, *and* $a \in V$,
*(g)* $(0, a \to i)$ *with* $1 \leq i \leq n$, *and* $a \in V$;

5. $c_I \in \{1, \ldots, n\}$ *is the* input cell.

Thus, a TPCU is defined as a collection of $n \geq 1$ cells denoted by $1, 2, \ldots, n$, each one of them containing a multiset over the given alphabet $V$. We also considers that the environment contains infinitely many copies of the objects in $E$ and initially it does not contain any object in $V \setminus E$. Cells are allowed to interact with each other and with the environment through the application of the rules in $R$. A rule $(i, a \to j)$ specifies that an object $a$ may be moved from cell $i$ to cell $j$ without any condition. A rule $(i, a \to j, b)$ with $1 \leq i, j \leq n$, $i \neq j$ and $a, b \in V$, specifies that if an object $b$ is in cell $j$, then an occurrence of symbol $a$ may be moved from cell $i$ to cell $j$. A rule $(i, b, a \to j)$, with $1 \leq i, j \leq n$, $i \neq j$ and $a, b \in V$, specifies that if an object $a$ and an object $b$ are both present inside cell $i$, then that object $a$ may be moved from cell $i$ to cell $j$. Similarly, in presence of an object $b$ inside cell $j$, a rule $(0, a \to j, b)$ may be used to move an occurrence of object $a$ from the environment, denoted by 0, to cell $j$; if an object $a$ and an object $b$ are both present inside cell $i$, then a rule $(i, b, a \to 0)$ may be used to move that object $a$ from cell $i$ to the environment. Rules $(i, a \to 0)$, $(i, a \to 0)$ can instead be used to move an object $a$ from cell $i$ to the environment and vice versa without any condition.

As usual in membrane computing, we adopt a non-deterministic maximal parallel strategy for the application of the rules which makes the system transit from one configuration to the other. Specifically, we have that, in each step, all the rules that can be applied, depending on the current distribution of objects inside the cells, must be applied. Objects are non-deterministically assigned to the rules with the only restriction that, within the same step, the same occurrence of the same symbol is used by at most one rule. This means that an occurrence of a symbol cannot be simultaneously moved and used to move another object, and can be used to move at most one occurrence of another symbol. However, the same rule can be used in parallel many different times to move many different objects. Moreover notice that rules do not modify the objects involved in their applications, hence, whenever a rule involves two objects, one is moved into some other cell whereas the other one is left untouched in the place where it is. We also remark the difference with respect to the more standard notion of promoters from [15]: promoters are multisets of objects that are used to activate a whole set of rules; the activated rules are then applied in a non-deterministic maximal parallel manner irrespectively of the multiplicity of the promoting multisets.

Let $\mathcal{T} = (V, E, w_1, \ldots, w_n, R, c_I)$ be a TPCU of degree $n \geq 1$. A configuration of $\mathcal{T}$ is any tuple $(w'_1, w'_2, \ldots, w'_n)$ with $w'i_i \in V^*$, for all $1 \leq i \leq n$. Then, given a multiset $x \in V^*$, TPCU $\mathcal{T}$ *recognises* multiset $x$ if, by starting from configuration $(w_1, \ldots, x\, w_I, \ldots, w_n)$, after a finite sequence of transitions obtained by applying the rules as described above, it produces a final configuration where no more rules can be applied to the objects placed inside the cells and the environment. Moreover, TPCU $\mathcal{T}$ recognises a family of multisets $M$ if, for all $x \in M$, $\mathcal{T}$

recognises $x$. If that is the case, we also say that TPCU $\mathcal{T}$ recognise the set of natural numbers $N(M) = \{ |x| \mid x \in M \}$.

Finally, we naturally associate to each TPCU a *communication graph* which represent the structure of the system as it is induced by the rules provided in the definition of the system.

**Definition 2.** *Let $\mathcal{T} = (V, E, C_1, \ldots, C_n, R, c_O)$ be a TPCU. The* communication graph of $\mathcal{T}$*, denoted by $\Gamma(\mathcal{T})$, is the undirected graph $(\{1, \ldots, n\}, A)$ where $\{i, j\} \in A$ if and only if, there is a rule $(i, a \to j, b)$ or $(i, b, a \to j)$ in $R$ for some $a, b \in O$.*

Thus, given a TPCU, its communication graph is the graph containing a node per each cell in the system and an edge between every two cells which are allowed to interact by means of some rule. As we will see, this notion is particularly useful to graphically represent the structure of a given TPCU.

In the next section, we introduce some macro-elements (*macros*, for short) that group several conditional uniport rules. Macros are seen as sub-functional units which can be combined to form larger "blocks of cells" dedicated to perform some specific tasks. Specifically, we will define macros necessary to construct blocks which simulate incrementing and decrementing instructions of a deterministic register machine. Thus, in Section 5, we will be able to show the computational completeness of TPCU's.

## 4 Macros

Here we present the details of the macros mentioned at the end of the previous section; we also introduce a specific graphical representation for them. We remark that the behaviour of each macro is non-deterministic, but instructions are grouped in such a way that macros have only one non-looping branch in the non-deterministic evolution. Then, after the description of these macros, we show how they can be combined to construct simulation blocks for the incrementing and decrementing instructions of a deterministic register machine.

### 4.1 Synchronisation of two signals

This macro, shortly denoted as syn2, is represented in the left part of Figure 1.

This macro aims to synchronise symbols $s_1$ and $s_2$ which are treated like signals. If both of them are present in input cells (1 and 2), then they will continue to output cells (3 and 4). More precisely, if object $s_1$ is present in cell 1 and object $s_2$ is present in cell 2, then object $s_1$ will go to cell 3 and object $s_2$ will go to cell 4. Notice that the opposite is true as well: if one of the two signals is missing, then the other one does not move. No assumption about the time necessary to do this operation is made, *i.e.*, it is not done in one time unit. We can only affirm that $s_1$ arrives in cell 3 before $s_2$ arrives in cell 4. We give below necessary rules that implement the syn2 macro.

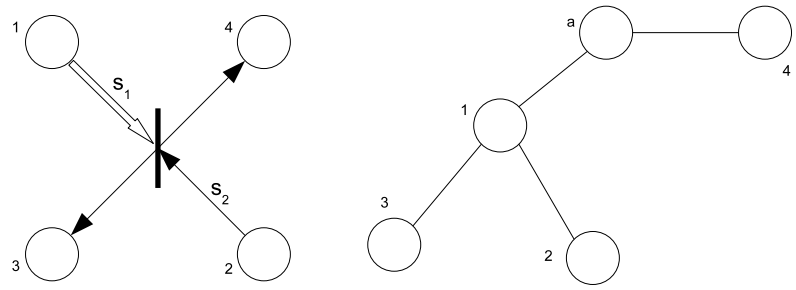


**Fig. 1.** syn2 element

1. $(2, s_2 \to 1, s_1)$
2. $(1, s_2, s_1 \to 3)$
3. $(a, X \to 1, s_2)$
4. $(a, X' \to 1, X)$
5. $(1, X' \to a, s_2)$
6. $(1, X \to a)$
7. $(1, X', s_2 \to a)$
8. $(a, X', s_2 \to 4)$

Symbols $X$ and $X'$ are initially present in cell $a$.

The communication graph induced by these rules is presented in the right part of Figure 1. It is easy to observe that the above structure, rules and initial objects permit to obtain the desired behaviour. Indeed, if symbol $s_1$ is present in cell 1 and there is no symbol $s_2$ in cell 2, then nothing happens. Similarly, if $s_2$ is present and $s_1$ is not present, this part of the system does not evolve. When both $s_1$ and $s_2$ are present, rule 1 brings $s_2$ to cell 1. After that either rule 2, or rule 3 will be applied (but not both of them because $s_2$ cannot be involved in more than one rule). Suppose that rule 3 is applied (the case of rule 2 is similar). In this case, symbol $X$ is brought to cell 1. At the next step, symbol $s_1$ is sent by rule 2 to cell 3, hence performing the first part of the synchronisation. At the same time one of the rules 4 or 6 is applied. If rule 6 is applied, then the system may loop using rules 3 and 6. Hence, at some moment rule 4 will be applied. This application brings symbol $X'$ in cell 1. This symbol permits to move $s_2$ to cell $a$ and further to cell 4. At the same time symbols $X$ and $X'$ return to their original place in cell $a$ and they are ready for another application of this macro-element.

We note that the synchronisation of $s_1$ and $s_2$ is the only non-looping evolution of the subsystem above. Moreover, the same group of cells of Figure 1 with the same communication graph may be reused for other pairs of signals by just adding similar rules for each pair of signals to be synchronised.

### 4.2 Synchronisation and duplication of two signals

This macro, shortly denoted as syndup, is represented in the left part of Figure 2.

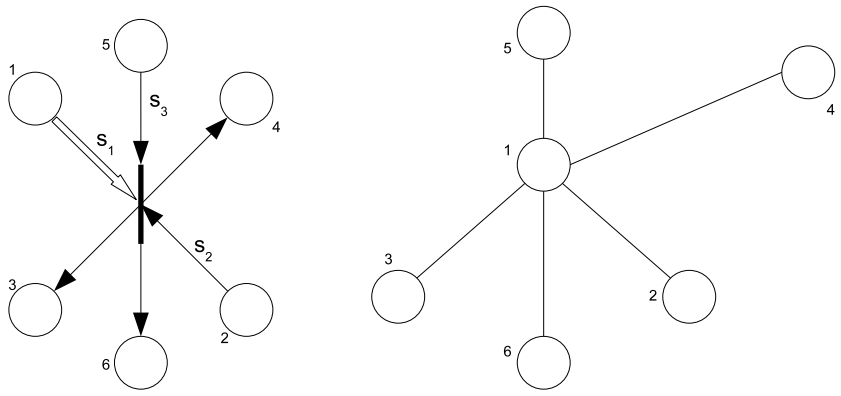


**Fig. 2.** syndup macro

This macro, like the previous one, synchronises symbols $s_1$ and $s_2$: if both of them are present in input cells (1 and 2), then they will continue to output cells (3 and 4). At the same time symbol $s_3$ goes from cell 5 to cell 6. If $s_2$ arrives to cell 4, it will be present there at least one step before $s_3$ arrives to cell 6. In some sense, $s_3$ is a copy of $s_2$ (which is a little bit time-shifted).

More precisely, if object $s_1$ (resp. $s_2$, $s_3$) is present in cell 1 (resp. cell 2, cell 5), then object $s_1$ will go to cell 3, object $s_3$ will go to cell 6 and object $s_2$ possibly will go to cell 4. If object $s_2$ is sent to cell 4, then it arrives there one step before $s_3$ arrives in cell 6. As before, no assumption about the time necessary to do this operation is made. We give below necessary rules that implement the syndup macro.

1. $(2, s_2 \to 1, s_1)$
2. $(1, s_2, s_1 \to 3)$
3. $(5, X \to 1, s_2)$
4. $(5, s_3 \to 1, X)$
5. $(1, X \to a)$
6. $(1, s_3, s_2 \to 4)$
7. $(1, s_3 \to 6)$

Symbol $X$ is initially present in cell 5.

The communication graph induced by these rules is presented in the right part of Figure 2.

Similarly to macro syn2, it is easy to observe that the above structure, rules and initial objects permit to obtain the desired behaviour. Symbol $s_1$ is sent to cell 3, while symbol $s_3$ may send symbol $s_2$ to cell 4. Finally, symbol $s_3$ goes to cell 6. There are two non-looping evolutions of this macro-element corresponding to the final position of $s_2$.

We remark that cells 1, 2 and 3 may be shared between syn2 and syndup macros.

### 4.3 Infinite loop

This macro, shortly denoted as i-loop, is represented in the left part of Figure 3.

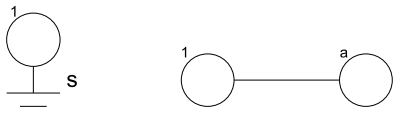


**Fig. 3.** i-loop macro

This macro has the following meaning. Object $s$ is always a subject to a computation in cell 1. Hence, if it is not taken from there, the system will never halt. We give below necessary rules that implement this macro.
1. $(1, s \to a, X_s)$
2. $(a, X_s, s \to 1)$

Symbol $X_s$ is initially present in cell $a$.

It is clear that above rules will infinitely move symbol $s$ between cells 1 and $a$.

### 4.4 Symbol check

This macro-instruction, shortly denoted as s-check, is represented in the left part of Figure 4.

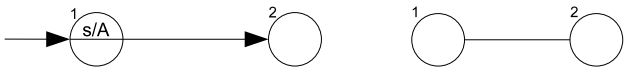


**Fig. 4.** s-check macro

This macro has the following meaning. In presence of object $A$ in cell 1, symbol $s$ from cell 1 will go to cell 2. Obviously, it corresponds to a rule $(1, A, s \to 2)$.

### 4.5 Incrementing and decrementing a counter

The left part of Figure 5 presents two macros denoted A-plus and A-minus.

These macros are used to increment/decrement the counter $A$. When object $s$ ($s_1$) arrives to cell 1, it increments (decrements) counter $A$ and after that $s$ ($s_1$) goes to cell 2. We give below necessary rules that implement these macros (cell 0 is the environment):

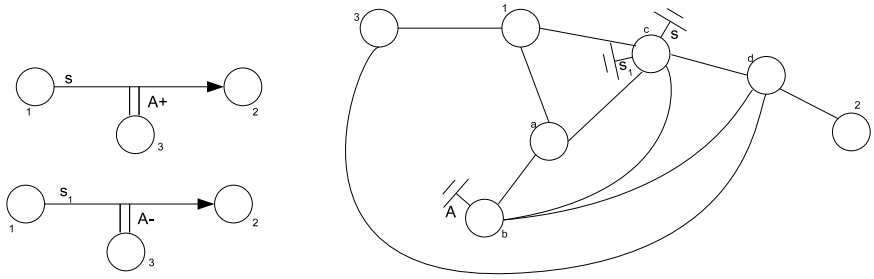


**Fig. 5.** A-plus and A-minus macros

1. $(3, A \to 1, s_1)$
2. $(0, A \to 1, s)$
3. $(1, s \to c)$
4. $(1, s_1 \to c)$
5. $(1, A \to a)$
6. $(a, A \to b)$
7. $(a, A \to c, s)$
8. $(a, A \to c, s_1)$
9. $(c, A \to b)$
10. $(c, A, s \to d)$
11. $(c, A, s \to d)$
12. $(c, A \to d, s)$
13. $(c, A \to d, s_1)$
14. $(d, A \to b)$
15. $(d, s \to 2)$
16. $(d, s_1 \to 2)$
17. $(d, s, A \to 3)$
18. $(d, s_1, A \to 0)$

The idea used to implement these macros is quite simple: symbol $s$ (resp. $s_1$) brings from the environment (resp. cell 3) symbol $A$ into cell 1. If more than one $A$ is brought, than at least one $A$ will arrive in cell $b$ triggering an infinite computation. Hence, the only possible halting computation is the one which brings an object $A$ from the environment, at the next step moves $A$ and $s$ ($s_1$) from cell 1 to cell $a$ and cell $c$ respectively, and then uses $s$ to move $A$ from cell $a$ to cell $c$. After that, in such a computation, both symbols are moved to cell $d$ , and finally $A$ will go to its place, while $s$ ($s_1$) reaches cell 2.

Notice that, although A-plus and A-minus are seen as separate macros, they are actually implemented through a unique group of cells containing rules for simulating both an incrementing and a decrementing instruction. In general, the same group of cells with the communication graph of Figure 5 can be used to simulate many different incrementing/decrementing instructions by joining the sets of rules necessary to simulate each instruction.

*Remark 1.* Macros defined above may be joined just by superposing some of their cells. Hence, a net-like structure may be built using these elements. Moreover, since we place objects in cells and after that move them, the obtained system has similarities with Petri Nets (e.g., see [16]). However, there are significant differences. In particular, macros above are in some sense time-less, because there is no limit on the number of steps necessary to perform their action. But in a halting configuration this will finally happen.

## 5 A Completeness Result

In this section we show how a register machine is simulated by using the macros introduced in the previous section, and hence we obtain the computational completeness of TPCU's. To this aim, we introduce the notation $NRTPCU_n$, for any $n \geq 1$, to denote the family of sets of natural numbers recognised by tissue P systems with conditional uniport.

**Theorem 1.** $NRTPCU_{24} = NRE$.

*Proof.* Let $M$ be a deterministic register machine. We will prove the assertion of the theorem in the following way. First, we construct $\Pi$ and we show that this system simulates the behaviour of $M$. In the same time, we investigate all other possible evolutions of $\Pi$ and we show that only evolutions corresponding to the simulation of $M$ lead to a halting configuration of $\Pi$.

In what follows we show how an arbitrary incrementing instruction $(p, A+, q)$ of $M$ is simulated. We mentioned before that we use macros defined in Section 4 as building blocks. We combine them as it is depicted in Figure 6. We number cells and we indicate their number below them. We remark that rules and objects of $\Pi$ can be easily deduced from these pictures.

The incrementing instruction is simulated as follows. Signals (symbols) $p$ and $A_{pq}$ synchronise, after that $A_{pq}$ increments register $A$, and synchronises with symbol $q$. After these actions $q$ is exchanged with $p$, $A_{pq}$ returns to its place and register $A$ is incremented.

As we have already said, all incrementing instructions of $M$ may be simulated using the same graph structure because macros for different instructions may share the same cells.

The case of a decrementing instruction $(p, A-, q, s)$ is depicted on Figure 7 (we remark that all working symbols ($D$, $s_1$, $s_2$) will be indexed by the number of the instruction). We number cells and we indicate their number below them. Specifically, in order to simulate a decrementing instruction, signals $p$ and $D$ ($D$ indeed stands for $D_{pqs}$) synchronise and if $D$ does not arrive in the output cell 6 (we recall that it arrives there non-deterministically), then $s_1$ will be kept in an infinite computation (in cell 9). Hence this branch will not halt. In the other branch of the computation, $D$ arrives in the output cell 6 and if there are symbols $A$ present, then it will move further. We remark that cell 6 stores counter symbols. Now, the symbol $s_1$, depending on position of $D$ will choose the corresponding branch of the computation. In this way the zero check on

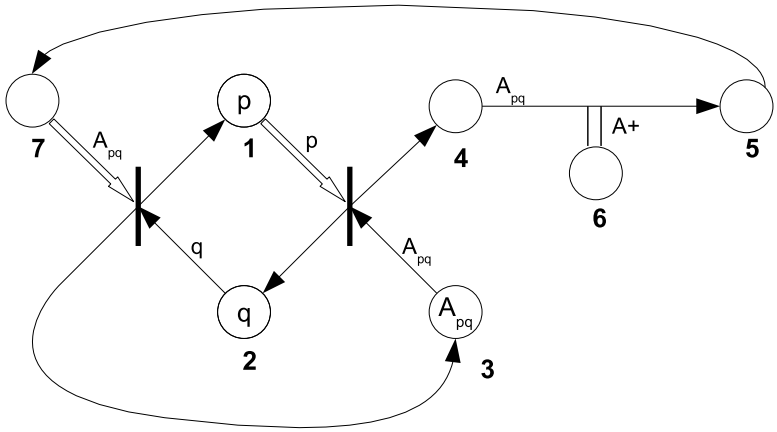


**Fig. 6.** Simulation of $(p, A+, q)$

register $A$ is performed. We note, that finally all additional symbols return to their original places. We remark that we use the same three cells (1, 2 and 7) for both signal synchronisations in the upper left corner.

Like in the previous case the simulation of all decrementing instructions may share the same graph. Moreover, the incrementing and incrementing graphical representations have a common underlying graph (which is in fact the graph corresponding to the decrementing enriched with some edges). The initial configuration contains the symbol $q_0$ at cell 1.

We stress once more that $\Pi$ can be easily deduced from the graphical representations corresponding to each instruction. From a structural point of view, cell 1 contains the current state, cell 2 contains all states of the machine except the current one. Cell 3 contains symbols $A_{pq}$ and $D_{pqs}$ that are used to drive the all simulation in $\Pi$ of corresponding instructions from $M$. Cell 6 contains unary values of all the registers. We also note that after expanding all macro-instructions, system $\Pi$ contains 24 cells.

For any configuration of $M$ $(q, A_1^{n_1}, \ldots, A_k^{n_k})$, there is a corresponding configuration of $\Pi$, where cell 1 contains $q$ and cell 6 contains symbols $A_1^{n_1}, \ldots, A_k^{n_k}$. Then, in order to finish the proof we need to show that the simulation of $M$ is the only possible halting computation. Indeed, we observe that the computation is started when a symbol $p$ corresponding to a state of $M$ arrives in cell 1 (initially in cell 1 symbol $q_0$ is present). Suppose, for simplicity, that there is the following instruction of $M$: $(p, A+, q)$. In this case, symbol $p$ goes to cell 2 where all unused state symbols are kept. In the meanwhile symbol $A_{pq}$ goes to cell 4 and drives the computation – first it increments register $A$ and after that synchronises with symbol $q$. At the end, symbol $A_{pq}$ returns to cell 2, while cell 1 contains $q$, hence corresponding instruction of $M$ is simulated.

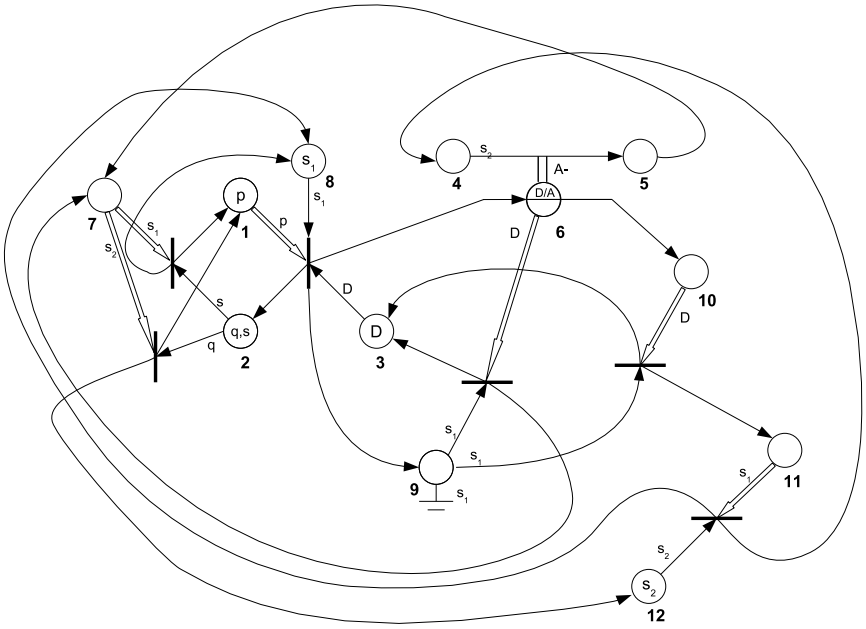


**Fig. 7.** Simulation of $(p, A-, q, s)$

For the decrementing case, symbol $p$ is synchronised with $D$ which drives the further computation. Therefore, at any moment, there is only one symbol that triggers the further computation (leading to a halting configuration). Since there is only one halting evolution, it corresponds to the simulation of $M$.

## 6 Discussion

In this paper a class of tissue P systems, called "with conditional uniport" or TPCU's, for short, is introduced. This model relies on simple communication rules which move simple symbol objects between adjacent components either freely or in the presence of another symbol object in one of these components. It is proved that this model, although with these very simple rules, it is computationally complete: it can recognise all recursively enumerable sets of natural numbers. In this respect, we also conjecture that, with some modifications in the proof of Theorem 1, TPCU's can be shown to be able to simulate non-deterministic register machines. This could lead to a characterisation of the family of recursively enumerable sets of natural numbers based on generative TPCU's which do not require an input multiset.

The idea of conditional communication is not completely new to the area of membrane computing. For instance, the concept of activated membrane channels was previously introduced in [7], and P automata [6] already considered

rules which allow a multiset to enter a membrane only in presence of another specific multiset inside that membrane. However, in these variants, "activators" are defined as generic multisets of any size, and, in order to achieve the power of Turing machines, activators of size at least two are always used. Thus, our approach is closer to minimal symport/antiport [17] as it reports completeness for systems with a "minimal" cooperation in the communication rules: activators consists of one single object, and rules involve at most two objects. On the other hand, with respect to results reported in [17], the completeness of conditional uniport is obtained by using a higher number of cells and tissue P systems with a complex graph structure. The optimality of this result is not known though, and this opens the possibility for improvements on the number of cells.

In addition to the completness result the paper introduces a set of "blocks" of tissue P system components using conditional uniport rules with a certain behaviour - the synchronisation of the objects moving between components, incrementing/decrementing the number of object symbols when a specific 'signal' object occurs in the block. These constructions are useful in order to simplify the proof of Theorem 1, but they might be considered as primitive components that (maybe with some restrictions or modifications) are combined to produce more powerful blocks. In our future works this approach will be investigated as a modular way of building solutions to some problems. This proposal is very relevant for investigations into modelling self-assembly phenomena by using P systems and their variants [5], [8] as it shows a great suitability in this respect.

## Acknowledgements

Marian Gheorghe and Francesco Bernardini are supported by the British Council research contract PN 05.014/2006, Maurice Margenstern and Sergey Verlan are funded by the Égide programme Alliance 0881UG. The authors are also grateful to the anonymous reviewers for their comments and suggestions that allowed us to improve this work.

## References

1. The P systems web page. `http://psystems.disco.unimib.it`.
2. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *The Molecular Biology of the Cell.* Garland Publ. Inc., London, 4th edition, 2002.
3. A. Alhazov, Y. Rogozhin, and S. Verlan. Tissue P systems with symport/antiport and minimal cooperation. to appear in *International Journal of Foundations of Computer Science*, 2006.
4. F. Bernardini and M. Gheorghe. Cell communication in tissue P systems: Universality results. *Soft Computing*, 9(9):640–649, 2005.
5. F. Bernardini, M. Gheorghe, N. Krasnogor, and J. L. Giavitto. On self-assembly in population P systems. In C.S Calude, M.J. Dinneen, Gh. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, editors, *Uncoventional Computation. 4th International Conference, UC 2005, Sevilla, Spain, October 2005, Proceedings*, volume 3365 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2005.

6. E. Csuhaj-Varjú and G. Vaszil. P automata or purely communicating accepting P systems. In Gh. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Argeş, Romania, August 19-23, 2002. Revised Papers*, volume 2597 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2003.
7. R. Freund and M. Oswald. P systems with activated/prohibited membrane channels. In Gh. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Argeş, Romania, August 19-23, 2002. Revised Papers*, volume 2597 of *Lecture Notes in Computer Science*, pages 261–269. Springer, 2003.
8. M. Gheorghe and Gh. Păun. Computing by self-assembly: DNA molecules, polynominoes, cells. In N. Krasnogor, S. Gustafson, D. Pelta, and J. L. Verdegay, editors, *Systems Self-Assembly: Multidisciplinary Snapshots*, Studies in Multidisciplinarity. Elsevier, 2005. In press.
9. E. Goles and M. Margenstern. Universality of the chip-firing game. *Theoretical Computer Science*, 172(1–2):91–120, 1997.
10. M. Margenstern. Two railway circuits: a universal circuit and an NP-difficult one. *Computer Science Journal of Moldova*, 9:3–33, 2001.
11. C. Martín-Vide, Gh. Păun, and G. Rozenberg. Membrane systems with carriers. *Theoretical Computer Science*, 270(1–2):779–796, 2002.
12. M. Minsky. *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
13. A. Păun and Gh. Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3):295–305, May 2002.
14. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
15. Gh. Păun. *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
16. W. Reisig. *Petri Nets. An Introduction*. Springer, Berlin, 1985.
17. Y. Rogozhin, A. Alhazov, and R. Freund. Computational power of symport/antiport: History, advances and open problems. In R. Freund, G. Lojka, M. Oswald, and Gh. Păun, editors, *Pre-Proceeding of the 6th International Workshop on Membrane Computing (WMC6)*, pages 44–78. TU Wien, Vienna, July 18-21, 2005.
18. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1–3. Springer, 1997.
19. T. Toffoli and N. Margolus. *Cellular automata machines*. The MIT Press, Cambridge, Mass., 1987.