School of Computing Science,
University of Newcastle upon Tyne

Towards a Petri net semantics for membrane systems

H.C.M. Kleijn, M.Koutny and G.Rozenberg

Technical Report Series

CS-TR-919

July 2005

Copyright©2005 University of Newcastle upon Tyne
Published by the University of Newcastle upon Tyne,
School of Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, UK.

Towards a Petri net semantics for membrane systems*

H.C.M.Kleijn[1], M.Koutny[2], and G.Rozenberg[1,3]

[1] LIACS, Leiden University
P.O.Box 9512, NL-2300 RA Leiden, The Netherlands
{kleijn,rozenber}@liacs.nl

[2] School of Computing Science, University of Newcastle,
Newcastle upon Tyne, NE1 7RU, United Kingdom
Maciej.Koutny@ncl.ac.uk

[3] Department of Computer Science, University of Colorado at Boulder,
Boulder, CO 80309-0347, USA

Abstract. We consider the modelling of the behaviour of membrane systems using Petri nets. First, a systematic, structural, link is established between a basic class of membrane systems and Petri nets. To capture the compartmentisation of membrane systems, localities are proposed as an extension of Petri nets. This leads to a locally maximal concurrency semantics for Petri nets. We indicate how processes for these nets could be defined which should be of use in order to describe what is actually going on during a computation of a membrane system.
Keywords: membrane systems, P systems, Petri nets, localities, causality and concurrency, processes.

1 Introduction

In the past 7 years *membrane systems*, also known as *P systems*, have received a lot of attention and in the process became a prominent new computational model [17–19, 1]. They are inspired by the compartmentisation of living cells and its effect on their functioning. A key structural notion is that of a *membrane* by which a system is divided into compartments where chemical reactions can take place. These reactions transform multisets of objects present in the compartments into new objects, possibly transferring objects to neighbouring compartments, including the environment. Consequently, the behavioural aspects of membrane systems are based on sets of *reaction rules* defined for each compartment. A distinguishing feature of membrane systems is that the system is assumed to evolve in a synchronous fashion, meaning that there is a global clock common for all the compartments. Within each time unit, the system is transformed by the rules which are applied in a maximally concurrent fashion (this means that no further rules in any compartment could have been applied

* Presented at the 6th Workshop on Membrane Computing, Vienna, July 18-21, 2005.

in the same time unit). These transformations are applied starting from an initial distribution of objects. Depending on the exact formalisation of the model, the notion of a successful (or halting) computation is defined together with its output, e.g., the number of objects sent to the environment.

The above describes the functionality of the basic membrane system model, according to [17, 19]. In addition, many different extensions and modifications of that basic model have been proposed and studied, such as priorities and catalysts. Moreover, those studies have been mostly focussed on the computational power of the models considered, including various aspects of complexity.

Given the existing body of results on the possible outcomes of computations of membrane systems, we feel that we are now in a position to also investigate and describe what is actually going on during a computation. The situation may be compared to that in the field of the semantics of programming languages based on input-output relations where the operational semantics was added to deal with the correctness of potentially non-terminating and concurrent programs. In this paper we propose to undertake this endeavour using the Petri net model (see, e.g., [21]). The reason is that they have local transformation rules and support the modelling of causality and concurrency in a direct and explicit way. In a nutshell, a Petri net is a bipartite directed graph consisting of two kinds of nodes, called *places* and *transitions*. Places together with their markings indicate the local availability of resources and thus can be used to represent objects in specific compartments, whereas transitions are actions which can occur depending on local conditions related to the availability of resources and thus can be used to represent reaction rules associated with specific compartments. When a transition occurs it consumes resources from its input places and produces items in its output places thus mimicking the effect of a reaction rule.

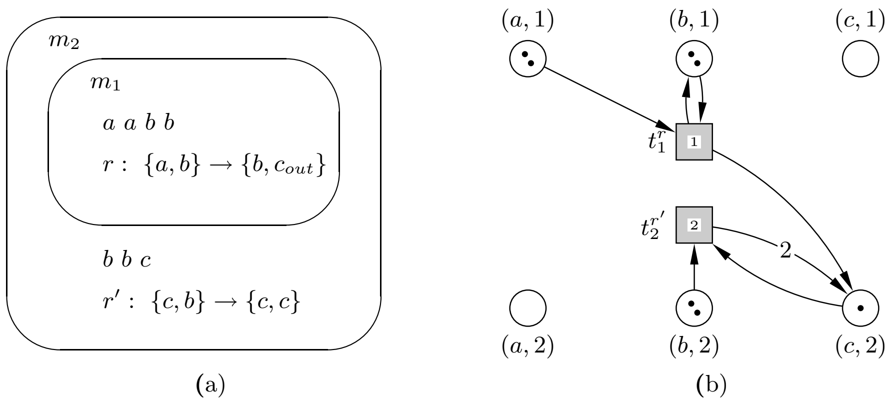

Fig. 1. A membrane system (a), and the corresponding Petri net (b).

The basic idea of modelling a membrane system using a Petri net can be explained through an example shown in Figure 1(a). The system depicted there consists of two nested membranes (the inner membrane m_1 and the outer membrane m_2), two rules (rule r associated with the compartment c_1 inside the inner membrane, and rule r' associated with the compartment c_2 surrounded by m_2, i.e., in-between the two membranes), and three symbols denoting molecules (a, b, and c). Initially, the compartment c_1 contains two copies of both a and b, and c_2 contains two copies of b and a single copy of c. To model this membrane system using a Petri net, we introduce a separate place (x, j) for each kind of molecule x and compartment c_j. As usual, places are drawn as circles with the number of the currently associated resources represented as tokens (small black dots). For each rule r associated with a compartment c_i we introduce a separate transition t_i^r, drawn as a rectangle. Transitions are connected to places by weighted directed arcs, and if no weight is shown it is by default equal to 1. If the transformation described by a rule r of compartment c_i consumes k copies of molecule x from compartment c_j, then we introduce a k weighted arc from place (x, j) to transition t_i^r, and similarly for molecules produced by transformations. Finally, assuming that initially compartment c_j contained n copies of molecule x, we introduce n tokens into place (x, j). The resulting Petri net is depicted in Figure 1(b). As argued later on, Petri nets derived in this way can be used to describe issues related to concurrency in the behaviour of the original membrane systems.

Applying Petri nets to model membrane systems is by no means an original idea. Since multiset calculus is basic for membrane systems and also for computing the token distribution in Petri nets [3], some connections have already been established. Some authors have in fact already proposed to interpret reaction rules of membrane systems using Petri net transitions, e.g., [5, 20]. Our aim is to demonstrate that a relationship between Petri nets and membrane systems can be established at the system level. We achieve this by defining a class of Petri nets suitable for the study of behavioural aspects of membrane systems and other systems exhibiting a mix of synchronous and asynchronous execution rules. This latter feature is motivated by the observation that the assumed strict global synchronicity of the membrane systems is not always reasonable from the biological point of view as already observed in [17]. In fact, [9] proposes to drop this assumption completely and considers fully asynchronous and sequential membrane systems; also the membrane systems of [5] are sequential, whereas [4] advocates that reactions are assigned their own execution times and uses a form of local synchronicity.

We intend to demonstrate that Petri nets obtained from membrane systems in the way described above provide a suitable model to capture and investigate the behavioural properties of membrane systems. In this sense the paper is more directed towards the computations taking place in membrane systems. After recalling the definition of membrane systems, we introduce a general class of Petri nets which can be used to define their formal concurrency semantics. This concurrency semantics will be built upon a well established technique of *unfolding* Petri nets, leading to *processes* which formalise concurrent execution histories. The paper deliberately avoids going into full technical details of the formal presentation, aiming instead at conveying the basic ideas of our proposal. Most of the formalities and proofs are delegated to the companion paper [14].

In this paper, a multiset (over a set X) is a function $\mathsf{m} : X \to \mathbb{N}$. By $\mathbb{N}^X$ we denote the set of multisets over X. For two multisets m and m' over X, we denote $\mathsf{m} \leq \mathsf{m}'$ if $\mathsf{m}(x) \leq \mathsf{m}'(x)$ for all $x \in X$. Moreover, a subset of X may be viewed through its characteristic function as a multiset over X, and for a multiset m we denote $x \in \mathsf{m}$ if $\mathsf{m}(x) \geq 1$. The sum of two multisets m and m' over X is given by $(\mathsf{m}+\mathsf{m}')(x) \overset{\mathrm{df}}{=} \mathsf{m}(x)+\mathsf{m}'(x)$, the difference by $(\mathsf{m}-\mathsf{m}')(x) \overset{\mathrm{df}}{=} \max\{0, \mathsf{m}(x)-\mathsf{m}'(x)\}$, as a total function extending set difference. The multiplication of m by a natural number n is given by $(n\cdot\mathsf{m})(x) \overset{\mathrm{df}}{=} n\cdot\mathsf{m}(x)$. Moreover, any finite sum $\mathsf{m}_1+\dots+\mathsf{m}_k$ will also be denoted as $\sum_{i\in\{1,\dots,k\}} \mathsf{m}_i$.

2 Basic membrane systems

For the purposes of this paper, it suffices to consider the most basic definition of membrane systems [18, 19]. Throughout the paper a *membrane system* (of degree $m \geq 1$) is a construct

$$\Pi \overset{\mathrm{df}}{=} (V, \mu, w_1^0, \dots, w_m^0, R_1, \dots, R_m)$$

where:

- V is a finite *alphabet* consisting of (names of) objects;
- μ is a *membrane structure* given by a rooted tree with m nodes, representing the membranes, as illustrated in Figure 2 — without loss of generality, we assume that the nodes are given as the integers $1, \dots, m$, and $(i, j) \in \mu$ will mean that there is an edge from i (parent) to j (child) in the tree of μ;
- each w_i^0 is a multiset of objects initially associated with membrane i;
- each R_i is a finite set of *evolution rules* r associated with membrane i, of the form:
$$lhs^r \to rhs^r$$
where lhs^r — the left hand side of r — is a non-empty multiset over V, and rhs^r — the right hand side of r — is a non-empty multiset over
$$V \cup \{a_{out} \mid a \in V\} \cup \{a_{in_j} \mid a \in V \textit{ and } (i,j) \in \mu\} \ .$$

Symbols a_{in_j} represent objects a that will be sent to a child node j and a_{out} stands for an a that will be sent out to the parent node. Without loss of generality,[1] we additionally assume that no evolution rule r associated with the root of the membrane structure uses any a_{out} in rhs^r.

[1] Since the environment can always be modelled by adding a new root to the membrane structure.

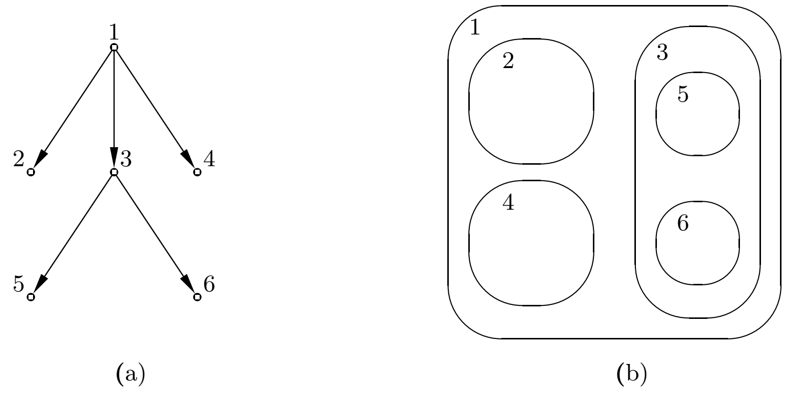

Fig. 2. A membrane structure (a); and the corresponding compartments (b).

A membrane system Π as above evolves from configuration to configuration as a consequence of the application of (multisets of) evolution rules in each membrane. Formally, a *configuration* is a tuple $C \overset{\text{df}}{=} (w_1, \ldots, w_m)$ where each w_i is a multiset of object names; we define a *vector multi-rule* $\boldsymbol{R}$ as an element of $\mathbb{N}^{R_1} \times \cdots \times \mathbb{N}^{R_m}$. Given a vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$, we use as additional notation $lhs_i = \sum_{r \in R_i} \widehat{R}_i(r) \cdot lhs^r$ for the multiset of all objects in the left hand sides of the rules in $\widehat{R}_i$ and, similarly, $rhs_i = \sum_{r \in R_i} \widehat{R}_i(r) \cdot rhs^r$ is the multiset of all — possibly indexed — objects in the right hand sides.

Given two configurations, $C = (w_1, \ldots, w_m)$ and $C' = (w'_1, \ldots, w'_m)$, C can *evolve* into C' if there exists a vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$ such that for every $1 \leq i \leq m$, the following hold

(i) $lhs_i \leq w_i$;
(ii) there is no rule r in R_i such that $lhs^r + lhs_i \leq w_i$; and
(iii) for each object $a \in V$,

$$w'_i(a) = w_i(a) - lhs_i(a) + rhs_i(a) + rhs_{parent(i)}(a_{in_i}) + \sum_{(i,j) \in \mu} rhs_j(a_{out}) \ ,$$

where $parent(i)$ is the father membrane of i unless i is the root in which case $parent(i)$ is undefined and $rhs_{parent(i)}(a_{in_i})$ is omitted. Note that any j in the last term must be a child membrane of i.

By (i), the configuration C has in each membrane i enough occurrences of objects for the application of the multiset of evolution rules $\widehat{R}_i$. Maximal concurrency is captured by (ii) according to which in none of the membranes an additional evolution rule can be applied. Observe that some of the $\widehat{R}_i$'s in $\boldsymbol{R}$ may be empty i.e., no evolution rules associated with the corresponding membranes i can be used. Finally, (iii) describes the effect of the application of the rules in $\boldsymbol{R}$.

By $C \overset{\boldsymbol{R}}{\Longrightarrow} C'$ we denote that C evolves into C' due to the application of $\boldsymbol{R}$. Note that the evolution of C is non-deterministic in the sense that there may be different vector multi-rules applicable to C as described above. A (finite) *computation* of Π is now a (finite) sequence of evolutions starting from the initial configuration $C_0 \overset{\text{df}}{=} (w_1^0, \dots, w_m^0)$.

3 Petri nets

We first recall the key notions of the standard Petri net model. A *PT-net* is a tuple $N \overset{\text{df}}{=} (P, T, W, M_0)$ such that P and T are finite disjoint sets; $W : (T \times P) \cup (P \times T) \to \mathbb{N}$ is a multiset; and M_0 is a multiset of places. The elements of P and T are respectively the *places* and *transitions*, W is the *weight function* of N, and M_0 is the *initial* marking. In diagrams, places are drawn as circles, and transitions as rectangles. If $W(x, y) \geq 1$ for some $(x, y) \in (T \times P) \cup (P \times T)$, then (x, y) is an *arc* leading from x to y. As usual, arcs are annotated with their weight if this is 2 or more. We assume that, for every $t \in T$, there are places p and q such that $W(p, t) \geq 1$ and $W(t, q) \geq 1$.

Places represent local states, while markings are global states of systems represented by PT-nets. Transitions represent actions which may occur at a given marking and then lead to a new marking (the weight function specifies what resources are consumed and produced during the execution of such actions).

Figure 3 shows a PT-net model of a simple one-producer / two-consumers concurrent system, where the producer is represented by the initial token in place p and the consumers by the two tokens in place r. Using transition `a`, the producer repeatedly produces new items (tokens) and *adds* them to place q (intuitively, a buffer between the producer and the two consumers) from where they can be *taken* by one of the two consumers, and then *used* by executing transition `u`. Rather than producing a new item, the producer may at any time *cancel* the production cycle by executing transition `c`.

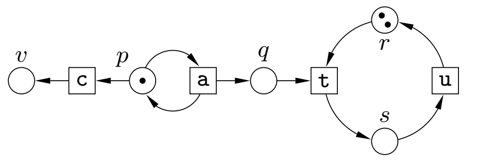

Fig. 3. PT-net of the one-producer / two-consumers system.

The *pre-* and *post-multiset* of a transition $t \in T$ are multisets of places given, for all $p \in P$, by:

$$\text{PRE}_N(t)(p) \overset{\text{df}}{=} W(p, t) \quad \text{and} \quad \text{POST}_N(t)(p) \overset{\text{df}}{=} W(t, p) \; .$$

Both notations extend to multisets of transitions U:

$$\textsc{pre}_N(U) \overset{\text{df}}{=} \sum_{t \in U} U(t) \cdot \textsc{pre}_N(t) \qquad \text{and} \qquad \textsc{post}_N(U) \overset{\text{df}}{=} \sum_{t \in U} U(t) \cdot \textsc{post}_N(t)\ .$$

A *step* is a multiset of transitions, $U : T \to \mathbb{N}$. It is *enabled* at a marking M if $M \geq \textsc{pre}_N(U)$. We denote this by $M[U\rangle$. Thus, in order for U to be enabled at M, for each place p, the number of tokens in p under M should at least be equal to the total number of tokens that are needed as an input to U, respecting the weights of the input arcs. Moreover, U is a *maximal step* at M if $M[U\rangle$ and there is no transition t such that $M[U + \{t\}\rangle$.

If U is enabled at M, then it can be *executed* leading to the marking $M' \overset{\text{df}}{=} M - \textsc{pre}_N(U) + \textsc{post}_N(U)$. This means that the execution of U 'consumes' from each place p exactly $W(p,t)$ tokens for each occurrence of a transition $t \in U$ that has p as an input place, and 'produces' in each place p exactly $W(t,p)$ tokens for each occurrence of a transition $t \in U$ with p as an output place. If the execution of U leads from M to M' we write $M[U\rangle M'$. Whenever U is a maximal step at M, we will also write $M[U\rangle_{max} M'$.

A finite sequence $\sigma = U_1 \ldots U_n$ of non-empty steps is a *step sequence* from the initial marking M_0 if there are markings $M_1 \ldots M_n$ of N satisfying $M_{i-1}[U_i\rangle M_i$ for every $i \leq n$. Such a σ is also called a step sequence from M_0 to M_n, and M_n itself is called a *reachable* marking.

In the same way, we can define step sequences consisting of maximal steps, and markings reachable through such step sequences. Together, they define the *maximal concurrency semantics* of the PT-net N as considered, for instance, in [6].

The example PT-net in Figure 3 admits an infinite number of step sequences. For example, $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ models the following scenario: (i) the producer produces an item which is then deposited into the buffer; (ii) the producer produces another item and, at the same time, one of the consumers takes the previously produced item from the buffer; and (iii) the consumer who retrieved the first item produced uses it and, at the same time, the second consumer removes the second item produced from the buffer. In Figure 4 we show how this scenario changes the current marking (global state) of the PT-net. As far as the maximal concurrency semantics is concerned, $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ is not allowed: though the first two steps executed are maximal, $\{\mathtt{u},\mathtt{t}\}$ is not since, for instance, the step $\{\mathtt{a},\mathtt{u},\mathtt{t}\}$ is enabled after the execution of $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$, and $\sigma' = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{a},\mathtt{u},\mathtt{t}\}$ rather than σ is part of the maximal concurrency semantics of the PT-net in Figure 3.

3.1 Petri nets with localities

In order to represent the compartmentisation of membrane systems we now introduce a novel extension of the basic net model of PT-nets, by adding the notion of located transitions and locally maximally concurrent executions of co-located transitions. In the proposed way of specifying locality for the transitions in a

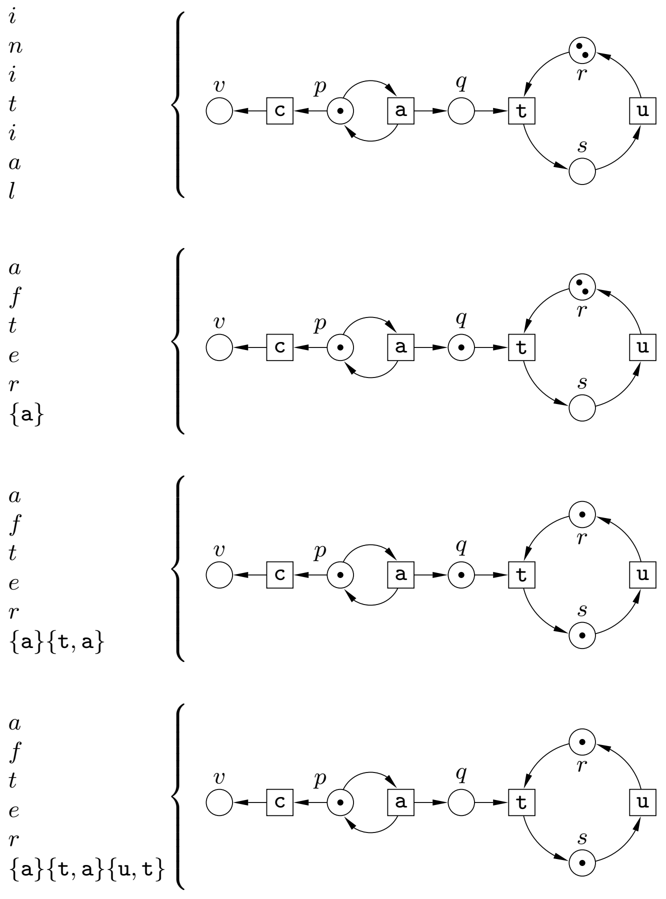

Fig. 4. Executing the PT-net according to {a}{t, a}{u, t}.

PT-net, each transition belongs to a fixed unique locality. The exact mechanism for achieving this is to introduce a partition of the set of all transitions, using a locality mapping $\mathfrak{L}$. Intuitively, two transitions for which $\mathfrak{L}$ returns the same value will be co-located.

A *PT-net with localities* (or PTL-net) is a tuple $NL \overset{\mathrm{df}}{=} (P, T, W, M_0, \mathfrak{L})$, where $\textsc{und}(NL) \overset{\mathrm{df}}{=} (P, T, W, M_0)$ is the *underlying* PT-net and $\mathfrak{L} : T \to \mathbb{N}$ is a *location mapping* for the transition set T. In the diagrams of PTL-nets, transitions are shaded rectangles with the locality being shown in the middle. Note that $\mathfrak{L}$ is merely a labelling of transitions, it is not meant as a renaming (as used later for occurrence nets).

The two execution semantics already defined for PT-nets carry over to PTL-nets, after assuming that all the notations concerning the places and transitions of a PTL-net are as in the underlying PT-net, together with the notions of marking, (maximal) step and the result of executing a step.

3.2 Membrane systems as Petri nets

In this section, we make our proposal on how membrane systems can be interpreted by Petri nets more precise. Given the definitions of membrane systems and Petri nets with localities, the construction sketched in the introduction can be implemented as follows.

Let $\Pi = (V, \mu, w_1^0, \dots, w_m^0, R_1, \dots, R_m)$ be a membrane system of degree m. Then the corresponding PTL-net is $NL_\Pi \overset{\text{df}}{=} (P, T, W, M_0, \mathfrak{L})$ where the various components are defined thus:

- $P \overset{\text{df}}{=} V \times \{1, \dots, m\}$;
- $T \overset{\text{df}}{=} T_1 \cup \ldots \cup T_m$ where each T_i contains a distinct transition t_i^r for every evolution rule $r \in R_i$;
- for every place $p = (a, j) \in P$ and every transition $t = t_i^r \in T$,

$$W(p,t) \overset{\text{df}}{=} \begin{cases} lhs^r(a) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad W(t,p) \overset{\text{df}}{=} \begin{cases} rhs^r(a) & \text{if } i = j \\ rhs^r(a_{out}) & \text{if } (j,i) \in \mu \\ rhs^r(a_{in_j}) & \text{if } (i,j) \in \mu \\ 0 & \text{otherwise} \end{cases}$$

- for every place $p = (a, j) \in P$, its initial marking is $M_0(p) \overset{\text{df}}{=} w_j(a)$.
- for every transition $t = t_i^r \in T$, its locality is $\mathfrak{L}(t) \overset{\text{df}}{=} i$.

To capture the very tight correspondence between the membrane system Π and the PTL-net NL_Π, we introduce a straightforward bijection between configurations of Π and markings of NL_Π, based on the correspondence of object locations and places.

Let $C = (w_1, \dots, w_m)$ be a configuration of Π. Then the corresponding marking $\phi(C)$ of NL_Π is given by $\phi(C)(a,i) \overset{\text{df}}{=} w_i(a)$, for every place (a,i) of NL_Π. Similarly, for any vector multi-rule $\boldsymbol{R} = (\widehat{R}_1, \dots, \widehat{R}_m)$ of Π, we define a multiset $\psi(\boldsymbol{R})$ of transitions of NL_Π such that $\psi(\boldsymbol{R})(t_i^r) \overset{\text{df}}{=} \widehat{R}_i(r)$ for every $t_i^r \in T$. It is clear that ϕ is a bijection from the configurations of Π to the markings of NL_Π, and that ψ is a bijection from vector multi-rules of Π to steps of NL_Π.

It should be clear that not every PTL-net can be obtained from a membrane system using the transformation described above. For example, in any net NL_Π, two transitions sharing an input place will always have the same locality assigned by $\mathfrak{L}$.

We now can formulate a fundamental property concerning the relationship between the dynamics of the membrane system Π and that of the corresponding PTL-net:

$$C \overset{\boldsymbol{R}}{\Longrightarrow} C' \quad \textit{if and only if} \quad \phi(C)\ [\psi(\boldsymbol{R})\rangle_{max}\ \phi(C')\ .$$

Since the initial configuration of Π corresponds through ϕ to the initial marking of NL_{Π}, the above immediately implies that the computations of Π coincide with the maximal concurrency semantics of the PTL net NL_{Π}.

The reader might by now have observed that the membrane structure of Π is used in the definitions of the static structure of the PTL-net NL_{Π} (i.e., in the definitions of places, transitions and the weight function), but as far as maximal concurrency semantics is concerned, the locality information for transitions in the form of the mapping $\mathfrak{L}$ of NL_{Π} is not relevant (the structure of Petri nets explicitly supports the locality aspects of the resources consumed and produced by transitions). However, it allows us to define local synchronicity presented next.

3.3 Locally maximal concurrency semantics of PTL-nets

Consider the PTL-model of the producer/consumer example as depicted in Figure 5. It conveys, in particular, the information that transitions `a` and `c` are assigned one locality, whereas transitions `t` and `u` are assigned another locality. This reflects the view that the producer operates away from the two consumers.

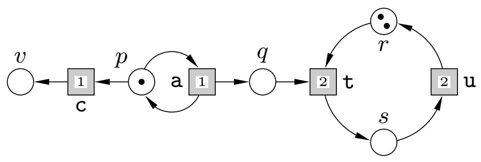

Fig. 5. PTL-net of the one-producer / two-consumers system.

To define a right semantical model reflecting this distribution of computing agents, we need to change the enabling condition for steps. Now, intuitively, only those steps are allowed to occur which are maximally concurrent within the localities given by $\mathfrak{L}$.

In a PTL-net $NL = (P, T, W, M_0, \mathfrak{L})$, a step $U : T \to \mathbb{N}$ is *locally max-enabled* at a marking M if it is enabled at M in $\textsc{und}(NL)$ and, in addition, there is no transition t such that $\mathfrak{L}(t) \in \mathfrak{L}(U)$ and $U+\{t\}$ is still enabled at M in $\textsc{und}(NL)$. Thus a step which is locally max-enabled at a marking is not necessarily a maximal step at that marking. The induced notions of a locally maximal step sequence and marking reachability are then defined as usual using the just defined notion of enabledness.

We now can look at the impact the various definitions of enabledness have on the set of legal behaviours of a Petri net. Looking at the PT-net N in Figure 3 and PTL-net NL in Figure 5, we can observe the following. First of all, the step sequence $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$, which was possible for N, is a legal behaviour of

NL under the locally maximal concurrency semantics as are many others, like $\{\mathtt{a}\}\{\mathtt{a}\}\{\mathtt{a}\}$ and $\{\mathtt{a}\}\{\mathtt{a}\}\{\mathtt{t},\mathtt{t}\}$. (Recall here that $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ was disallowed by the maximal concurrency semantics.) However, there are also step sequences of N which are not part of the locally maximal concurrency semantics of *NL*; e.g., $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{t}\}$ since after executing $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$ it is possible to execute step $\{\mathtt{u},\mathtt{t}\}$ which is strictly greater than $\{\mathtt{t}\}$ and transitions $\mathtt{t}$ and $\mathtt{u}$ are co-located.

Coming back to the example shown in Figure 1(b), we have the following step sequences in the maximal concurrency semantics: the empty sequence, $\{t_1^r, t_1^r, t_2^{r'}\}$ and $\{t_1^r, t_1^r, t_2^{r'}\}\{t_2^{r'}\}$. The locally maximal concurrency semantics, on the other hand, yields several additional step sequences, like $\{t_1^r, t_1^r\}\{t_2^{r'}, t_2^{r'}\}$ and $\{t_2^{r'}\}\{t_1^r, t_1^r\}\{t_2^{r'}\}$. Note further that it does not allow $\{t_1^r, t_1^r\}\{t_2^{r'}\}$ which, in turn, is allowed by the standard step sequence semantics.

To summarise, PT-nets admit both standard and maximal concurrency semantics, while for PTL-nets we have in addition locally maximal concurrency semantics. In particular, this means that we cannot identify the exact semantical model just by looking at a net's structure; we always need to specify which execution semantics is being used.

4 Causality and concurrency

All three variants of step sequence semantics of a Petri net considered in this paper provide important insights into the concurrency aspects of the underlying systems. They are, however, still sequential in nature in the sense that steps occur ordered thus obscuring the true causal relationships between the occurrences of transitions. On the other hand, information on causal relationship is often of high importance for system analysis and/or design. Petri nets can easily support a formal approach where this information is readily available as was recognised a long time ago, see [16] where it was proposed to unfold behaviours into structures allowing an explicit representation of causality, conflict and concurrency. A well-established way of developing such a semantics for the standard PT-nets is based on a class of acyclic Petri nets, called *occurrence nets* [22]. What one essentially tries to achieve is to trace the changes of markings due to transitions being executed along some legal behaviour of the original PT-net, and in doing so record which resources were consumed and produced.

In this section, we first explain the main ideas behind the causality semantics based on standard step sequences of PT-nets. After that, we show how this approach could be adapted to work for the locally maximal concurrency semantics of PTL-nets. Note that the maximal concurrency semantics of a PT-net coincides with the locally maximal concurrency semantics of this PT-net after extending it to a PTL-net with all transitions mapped to the same locality; hence we will only consider explicitly the locally maximal concurrency semantics.

4.1 Causal behaviours of PT-nets

Looking at the sequence $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ of executions in Figure 4, it is not immediate that transition $\mathtt{u}$ could have occurred before the second occurrence of transition a or, in other words, that the former is not causally dependent on the latter.

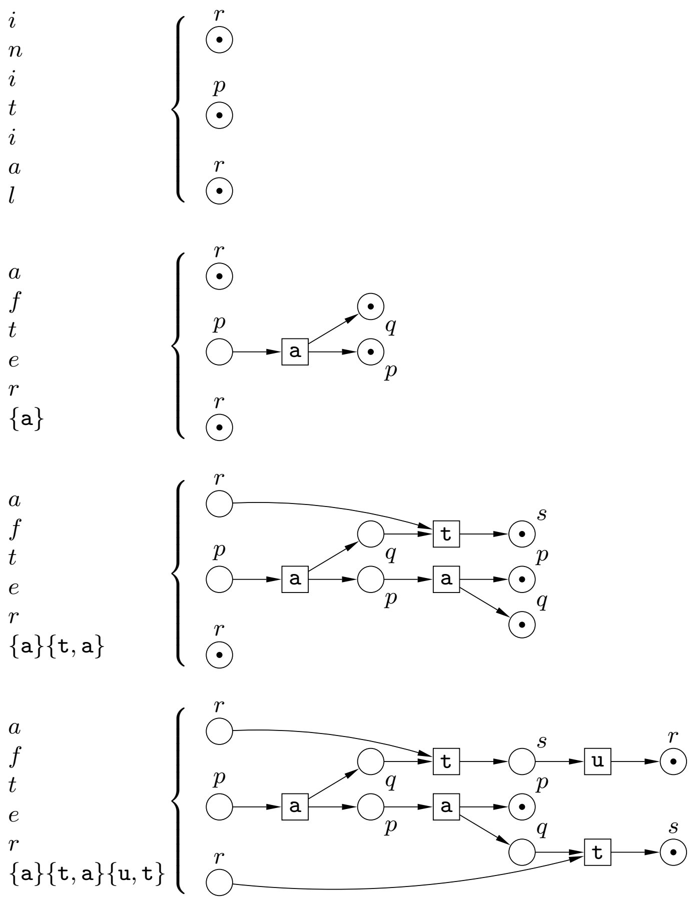

Fig. 6. Constructing an occurrence net corresponding to {a}{t, a}{u, t}.

Figure 6 illustrates the idea in which we *unfold* the scenario represented by σ. The initial stage shows just the initial marking which includes two separate (labelled) *conditions* (this is how places are called in occurrence nets) to represent the two initial tokens in place r. Executing step {a} consumes the p-condition, creates an a-event (this is how transitions are called in occurrence nets), as well as two new conditions: a p-condition and a q-condition. An important point is to notice that we create a fresh p-condition rather than a loop back to the initial one since we want to distinguish between different occurrences of the same token; as a result the occurrence net being constructed will be an acyclic graph. Another important point is that the environment of the generated a-event corresponds exactly to the environment of transition a; namely, it consumes a p-token and creates a p-token and a q-token. After that, executing step $\{\mathtt{t},\mathtt{a}\}$ consists in consuming three conditions and creating two events and three fresh conditions, and similarly for the last step $\{\mathtt{u},\mathtt{t}\}$. And, as a final result, we obtain an acyclic net labelled with places and transitions of the original PT-net; it is called a *process* of the original PT-net. The process net has a default initial marking consisting of a token in each of the conditions without an incoming arc.

It is now possible to look both at the structure of the process net and the executions which are possible from its default initial marking, making some important observations relating to:

– *Causality.* The causality relationships among the executed transitions can be read-off by following directed paths between the events; for example in Figure 6, the lower t-event is caused by both a-events, while the upper one is caused only by the leftmost a-event.
– *Concurrency.* Events for which there is no directed path from one to another can be thought of as concurrent.
– *Reachability.* Any maximal set of conditions for which there is no directed path from one condition to another corresponds to a reachable marking of the original PT-net.
– *Representation.* The step sequence on the basis of which the process was created can be executed from the initial default marking in the occurrence net. So the original behaviour has been retained. In Figure 6, there are 13 different step sequences generated by the process net defined by $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$, including σ itself.
– *Soundness.* Any step sequence which can be executed from the default initial marking to the default final marking (consisting of tokens placed in each of the conditions without an outgoing arc) of the process net is also a legal step sequence of the original PT-net. Processes provide a highly compressed representation of step sequence behaviours of the original PT-net (this feature has been exploited to a significant degree in the development of efficient model checking algorithms for PT-nets).

The above advantages of the process nets of the standard PT-nets lead us to consider a similar treatment for the PTL-net model and their locally maximal concurrency semantics.

4.2 Causal behaviours of PTL-nets

As a first attempt, we simply adopt the unfolding strategy as in the PT-net case. We only ensure that the step sequence consists of (locally) maximal steps. Moreover, we preserve the localities of the transitions in the events created while constructing the occurrence net. Figure 7 shows the result for the PTL-net of Figure 5 and the step sequence $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$ which is allowed in the maximal and thus also in the locally maximal concurrency semantics (both the occurrence net and its default initial marking are depicted). Although this is straightforward, we still need an argument that the resulting process is what one would want to take for further analyses. In particular, one would want to retain the soundness of the previous construction. In the case of our example, we can execute the occurrence net and conclude that under the maximal rule it admits the original sequence, whereas under the locally maximal rule it admits two more step sequences, {a}{a}{t} and {a}{t}{a}. And, clearly, both are legal step sequences of the original PTL-net in the locally maximal concurrency semantics.

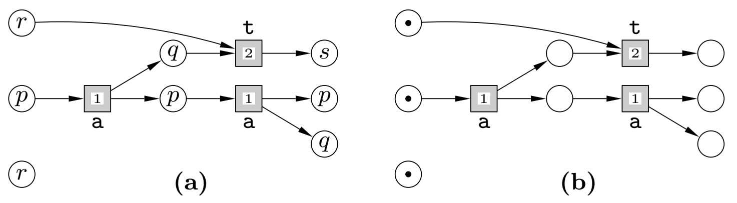

Fig. 7. Process net corresponding to the step sequence {a}{t, a} **(a)**; and its default initial marking **(b)**.

In general, however, it would be too hasty to accept the standard unfolding routine as satisfactory. Consider, for example, the PTL-net in Figure 8(a) and its step sequence {t, u, v}{w, z} consisting of locally maximal steps. Proceeding as in the previous case, we obtain an occurrence net shown in Figure 8(b). And the problem is that it has an execution from the default initial marking (using only locally maximal steps) which corresponds to {u, v}{t, z}{w}. This step sequence, however, is not a locally maximal step sequence of the original PTL-net as in the second step it is possible to add transition x which is co-located with transition z.

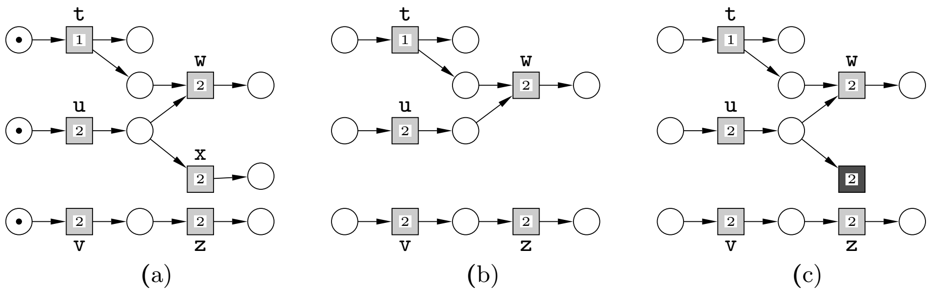

Fig. 8. PTL-net **(a)**; an occurrence net constructed from step sequence {t, u, v}{w, z} **(b)**; and a barbed process **(c)**.

An intuitive reason why the standard construction fails to work for the PTL-net in Figure 8(a) is that such an unfolding 'forgets' that transition x was enabled at a stage where transition w was selected. Then, delaying the execution of the `w`-event, creates a situation where the executed step (though locally maximal within the occurrence net since the knowledge of the enabledness of `x` is lost) does not correspond to a locally maximal step within the PTL-net.

Our approach to cope with this problem in [14] is to equip occurrence nets generated by PTL-nets with additional *barb-events*, represented by darkly shaded rectangles. Barb-events are not labelled with transition names and are not meant to be executed; rather, they are used in the calculation of the enabled sets of events. Such occurrence nets are called *barbed* processes. Rather than providing a full formal definition of how barb-events are added during the unfolding procedure, which we give in the companion paper [14], we only mention here that it is based on checking that transitions have not been included in the executed scenario since another co-located transition was selected. Figure 8(c) illustrates the modified construction for the net in Figure 8(a,b).

After executing $\{\mathtt{u},\mathtt{v}\}$, it is now impossible to select $\{\mathtt{t},\mathtt{z}\}$ since there is a record in the form of the barb-event that such a step would not be maximal in the locality to which transition $\{\mathtt{z}\}$ belongs. The only way of continuing is to execute $\{\mathtt{t}\}$ and after that $\{\mathtt{z},\mathtt{w}\}$, generating a legal step sequence $\{\mathtt{u},\mathtt{v}\}\{\mathtt{t}\}\{\mathtt{z},\mathtt{w}\}$.

5 Summary and conclusions

In this paper we have proposed an approach to the modelling of the behaviour of membrane systems through a class of Petri nets with localities (PTL-nets).

We gave first a formal translation for a basic class of membrane systems, and argued that the structure of the (maximally concurrent) computations of such membrane systems is faithfully reflected by the maximal concurrency semantics of the corresponding PTL-nets. This corresponds to the situation whereby all the rules are governed by a single global clock which corresponds to the case of maximally concurrent executions, as investigated in [6]. Hence the results on the reachability of certain markings (or, equivalently, configurations in membrane systems) developed there could form the basis of an investigation, e.g., whether a particular combination of molecules in certain compartments can happen in the legal evolutions of a membrane system.

After that we moved to a less centralised view of concurrent executions, as already advocated e.g., in [9], and defined a locally maximal concurrency semantics for PTL-nets. However, in case of individual localities for all transitions, we are not exactly dealing with the asynchronous or sequential systems, proposed by [9]. Since we maintain the requirement of locally maximal concurrency executions, the resulting systems exhibit maximal *autoconcurrency.*

In the model of PTL-nets there are no additional requirements on the relationship between transitions and their localities; in particular, as already mentioned, transitions with shared input places do not have to be co-located. Moreover, the flow of resources among the localities does not necessarily follow a tree-like structure. In fact, PTL-nets with their locally maximal concurrency semantics constitute a very general framework in which membrane systems and even conglomerates of membrane systems (organisms) can be expressed and studied.

An important feature characterising the proposed basic PTL-net model is its robustness, in the sense of being easily extendable to handle salient features of more sophisticated membrane systems. Examples of such features are: (i) priorities among rules which can be dealt with using Petri nets with priorities, e.g., as in [2]; (ii) catalysts governing the enabling of the reaction rules purely by their presence which can be dealt with using Petri nets with read arcs, e.g., as in [24]; (iii) substances forbidding certain reactions which can be dealt with using Petri nets with inhibitor arcs, e.g., as in [13]; and (iv) dissolution of membranes which can be dealt with using Petri nets with transfer arcs; e.g., as in [23, 7]. We could also consider membrane systems with rules having variable discrete durations, by suitably exploiting the locally maximal concurrency semantics of PTL-nets. Further investigation is also needed into the relationship between various P systems and a wide variety of restricted/extended Petri nets, such as [10, 11].

We finally outlined how a causality based semantics of PTL-nets could be defined and used to analyse the intricate details of concurrent computations of membrane systems. The proposed semantics is based on the unfolding of PTL-nets with the novel feature of *barb-events* needed to reflect choices in the locally maximal executions. Among the potential benefits of the proposed unfolding-based semantics is the efficient model checking approach to the verification of properties of concurrent systems modelled as Petri nets [8, 15, 12].

Summarising, we have developed a new systematic link between Petri nets and membrane systems which (hopefully) is useful for both areas. We see this formalization only as a beginning of the research into the representation of the behaviour of membrane systems through concurrent processes.

Clearly, one could simply use the basic model of PT-nets and simulate by 'brute force' the behaviour of membrane systems. In general, however, a biologist's interest will be in how a system functions and not just in what is delivered at the end. From the modelling point of view it is therefore more convenient to include localities as a direct interpretation of 'where is what'. This also provides the possibility to introduce a notion of local synchronicity as opposed to a global clock governing the evolution of a system. The process semantics of PT-nets provides an additional formal tool to study how a system functions rather than what it computes. Whereas step sequences can be viewed as ordered by a clock, processes can be used to represent causalities. Moreover using (infinite) processes, also ongoing (potentially infinite) system behaviour can be investigated, which is also interesting from a biological point of view.

For PT-nets the notion of locality inspired by membrane systems is a new interesting feature. The process semantics for PTL-nets working under the (locally) maximal concurrency semantics still has to be developed. In this paper we have briefly indicated how the technical problems could be solved. In addition, a proper notion of causality (order relation) based on processes (see the semantical scheme of [13]) and relevant for the biologically motivated membrane systems has to be identified as well.

Acknowledgment The authors are grateful to Hendrik Jan Hoogeboom for his comments on an earlier version of this paper. This research was supported by the EPSRC project CASINO.

References

1. Membrane systems web page: `http://psystems.disco.unimib.it/`
2. E.Best and M.Koutny (1992). Petri Net Semantics of Priority Systems. *Theoretical Computer Science* 96, 175–215.
3. C.S.Calude, Gh.Păun, G.Rozenberg and A.Salomaa (Eds.) (2001). *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View.* Springer-Verlag, Lecture Notes in Computer Science 2235.
4. M.Cavaliere and D.Sburlan (2005). Time-Independent P Systems. Proc. of *WMC 2004*, G.Mauri et al. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 3365, 239–258.
5. S.Dal Zilio and E.Formenti (2004). On the Dynamics of PB Systems: a Petri Net View. Proc. of *WMC 2003*, C.Martín-Vide et al. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2933, 153–167.
6. R.Devillers, R.Janicki, M.Koutny and P.E.Lauer (1986). Concurrent and Maximally Concurrent Evolution of Nonsequential Systems. *Theoretical Computer Science* 43, 213–238.
7. C.Dufourd (1998). *Réseaux de Petri avec Reset/Transfert: Décidabilité et Indécidabilité.* ENS Cachan.
8. J.Esparza, S.Römer and W.Vogler (2002). An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design* 20(3), 285–310.
9. R.Freund (2005). Asynchronous P Systems and P Systems Working in the Sequential Mode. Proc. of *WMC 2004*, G.Mauri et al. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 3365, 36–62.
10. O.H.Ibarra, Z.Dang and O.Egecioglu (2004). Catalytic P Systems, Semilinear Sets, and Vector Addition Systems. *Theoretical Computer Science* 312(2-3), 379–399.
11. O.H.Ibarra, H.-C.Ye and Z.Dang (2004). The Power of Maximal Parallelism in P Systems. Proc. of *DLT 2004*, C.Calude et al. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 3340, 212–224.
12. V.Khomenko, M.Koutny and W.Vogler (2003). Canonical Prefixes of Petri Net Unfoldings. *Acta Informatica* 40, 95–118.
13. H.C.M.Kleijn and M.Koutny (2004). Process Semantics of General Inhibitor Nets. *Information and Computation* 190, 18–69.
14. H.C.M.Kleijn, M.Koutny and G.Rozenberg (2005). *Process Semantics of Petri Nets with Localities.* In preparation.
15. K.L.McMillan (1992). Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. Proc. of *CAV'1992*, G.von Bochmann and D.K.Probst (Eds.). Springer-Verlag, Lecture Notes in Computer Science 663, 164–174.
16. M.Nielsen, G.Plotkin and G.Winskel (1980). Petri Nets, Event Structures and Domains, Part I. *Theoretical Computer Science* 13, 85–108.

17. Gh.Păun (2002). *Membrane Computing, An Introduction*. Springer-Verlag.
18. Gh.Păun (1999). Computing with Membranes. An Introduction. *Bulletin of the EATCS* 67, 139–152.
19. Gh.Păun and G.Rozenberg (2002). A Guide to Membrane Computing. *Theoretical Computer Science* 287, 73–100.
20. Z.Qi, J.You and H.Mao (2004). P Systems and Petri Nets. Proc. of *WMC 2003*, C.Martín-Vide et al. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2933, 286–303.
21. W.Reisig and G.Rozenberg (Eds.) (1998). *Lectures on Petri Nets*. Springer-Verlag, Lecture Notes in Computer Science 1491,1492.
22. G.Rozenberg and J.Engelfriet (1998). *Elementary Net Systems*. In: Lectures on Petri Nets I: Basic Models. Springer-Verlag, Lecture Notes in Computer Science 1491, 12–121.
23. R.Valk (1978). Self-Modifying Nets, a Natural Extension of Petri Nets. Proc. of *ICALP 1978*, G. Ausiello and C.Böhm (Eds.). Springer-Verlag, Lecture Notes in Computer Science 62, 464–476.
24. W.Vogler (2002). Partial Order Semantics and Read Arcs. *Theoretical Computer Science* 286, 33–63.