Reversing Steps in Membrane Systems Computations

G. Michele Pinna(✉)

Dipartimento di Matematica e Informatica, Università di Cagliari, Cagliari, Italy
gmpinna@unica.it

Abstract. The issue of reversibility in computational paradigms has gained interest in recent years. In this paper we investigate how to *reverse* steps in membrane systems computations. The problem is that computation steps in membrane systems do not preserve all the information that has to be used when reversing them. We try to formalize the relevant information needed, and we show that the proposed approach enjoy the so called *loop lemma*, which basically assures that the undoing obtained by reversely applying rules is correct.

1 Introduction

Membrane systems, introduced by Păun (see [22,23] for a first account on membrane systems), are nowadays a popular and extensively studied computational paradigm inspired by how computations in the living cells take place.

The ingredients of this computational paradigm are a *membrane structure* (which is a tree-like structure), a multiset of *objects* associated to each membrane (spatial distribution of resources) and a set of *evolution rules* for each membrane (acting at local states). A computation step is performed by the application of a bunch of rules which consume objects from a membrane and produce objects in this membrane and possibly in the neighbouring membranes as well. All the possible instances of applicable rules are used, as it happens usually in nature, but this is not actually always needed to make the computational paradigm Turing equivalent (for instance, in [21] membrane systems where the rules have a special format are considered, and maximality is not required; similarly in [5] or [9] where, respectively, minimal parallelism or the presence of special objects called *catalysts* is considered). The computational paradigm has also been compared with many other paradigms (see [23] and the chapters therein for a fairly detailed account), and the relative expressivity has been studied (see, for instance, [6,7] or [4]).

Reversibility in computation paradigms is an issue that recently has received great attention[1]. Reversibility in nature has a quite precise meaning: once

Work partially supported by RAS, Sardinia Regional Government, Convenzione triennale tra la Fondazione di Sardegna e gli Atenei Sardi Regione Sardegna, (CUP: F72F16003030002), and P.I.A. 2013 "NOMAD".

[1] This is testified by the series of workshops and conferences entitled *Reversible Computation* (RC) organized since 2009, which is a conference since 2013.

reached a certain state (say F) from a starting one (say I) with a sequence of steps, there is the capability of reaching again the state I, possibly applying the various steps in reverse order. Furthermore in nature also energy is considered, and the final balance after reversing steps should be zero. When focussing on computational devices, reversibility in general accounts on understanding how certain rules are applicable in reverse order and in particular the amount of information to be preserved. We do not discuss here further why reversibility is worth to be considered, and we refer to [16] for more motivations.

As already pointed out in [16], when reversibility is *backtracking* (the feature that certain *non deterministic* computations enjoy, which allows to *explore* all the possible alternatives), then reversibility is well understood. Just a suitable coding of the choices and of the applied rules is enough. It is much less clear in the case of *distributed* or *concurrent* systems, where the applications of the rules is done in a local fashion. Membrane systems have to be considered computing devices of this kind.

The aim of this paper is to investigate on how to *reverse* computation steps in membrane systems in such a way that if a configuration is reached from another one, there is a way to reverse this step.

Reversing computations can be achieved in various manners. One is to add suitable rules having the *reverse* effect of other rules, another approach is orthogonal to this one and it is based on devising on how to apply the same rule *reversely*. In the first way the fact that a computation is reversed is just a matter of observation on the results achieved by the computation itself, making the approach a sort of *simulation* of reversibility. Still some information about the computations may have to be considered (for instance, in the approach presented in [1] some information should be kept, as rules having as effect the dissolution of membranes are considered). In the case it is necessary to keep track of the previous existence of some membranes.

Another way focusses on how a rule is *reversely* applied to a given stage of the computation, and to apply a rule reversely it is often necessary to keep some information about the previous stages of the computation. The conditions to be established are such that a *loop lemma* can be proven. The loop lemma simply states that if one goes from a stage of the computation to another one by applying a bunch of rules together, then from the reached stage it is possible to *go back* by reversely applying the same bunch of rules. Though this seems to be an easy and minimal requirement, it is not obvious that it generally holds for concurrent and distributed systems. In fact, as discussed in [8] (see also [16]), further information have to be given in order to be able to reverse computations steps consistently. The added information have to guarantee that the previous stage of the computation can be *reconstructed* properly.

We achieve this result by enriching the notion of configuration of a membrane system with a *memory* which records the (minimal) information to be considered in reversing steps. The notion of memory we adopt is similar to the one of event structure associated to a membrane system developed in [2,20]. We are able to prove the loop lemma, though we get a weaker version which we will discuss.

The choice of adding a memory is not the unique solution to the problems posed when reversing steps, as objects may be enriched with the full history. However the amount of information the memory may have is able to cover this other approach, hence we believe that what we propose is general enough.

Though reversibility often means to *fully* undo some steps, it is important to observe that this is not actually needed. In fact it seems more reasonable to allow to undo part of the steps rather than the whole one and this is feasible in our approach.

The paper in organized as follows. In the remaining part of this introduction we briefly recall how the issue of reversibility in membrane systems has been considered in literature.

In the next section we give some background and in Sect. 3 we review the notion of membrane systems and formalize the notion of membrane systems computation. In Sect. 4 we first state in general what reversing computations in membrane systems may be, discussing briefly its limitation, and then develop our approach: in Subsect. 4.1 we discuss how the information is added to configurations. Few ideas for future developments conclude the paper.

Reversibility in membrane systems: other approaches. Reversibility has been previously considered in membrane systems. In [1] Agrigoroaiei and Ciobanu present a first attempt to study reversibility in membrane systems. They develop a way to consider new *reversed* rules, called *dual* rules. Dual rules replace the original rules of the membrane system and reversed computations are studied with the aim at easing the search of appropriate solutions to problems *backward* rather than *forward* (and indeed the membrane systems introduced and studied in [1] are also called *dual* membrane systems). Thus computations are reversed as the whole system is actually reversed, which is different from undoing something.

Other papers consider when computations can be reversed, and they usually require that the membrane systems looked at are deterministic. For instance, [3] considers membrane systems where reversibility (or strong reversibility) means that every reachable configuration of the system can be *obtained* by a single configuration (and in the stronger version the reachability request is dropped), and the determinism issue is considered as well, meaning that every reachable configuration has just one successor configuration (again the strong version is the one requiring that the reachability request is dropped). In this paper conditions to achieve reversibility, strong reversibility, determinism and strong determinism are studied, and the expressivity of the associated systems is made precise. In [11] the problem of strong reversibility is further studied, and it is shown that it is decidable if a membrane system is strongly reversible. It is also worth to stress that the membrane systems considered have just one membrane.

In [18] the author considers membrane systems with *symport/antiport* rules, and it is shown that every reversible register machine can be simulated by a *deterministic* membrane system with *symport/antiport* rules. In [24] spiking neural systems are taken into account, and the investigations focus on the expressivity issue. Indeed it is shown that these systems are reversible because they are equivalent to reversible computing machines, and in all the above mentioned approaches the focus is on deterministic systems, whereas we consider reversibility without constraining it to special cases.

In [17] reversibility is considered as it is shown how to simulate Fredkin circuits with membrane systems, focussing on energy. Being Fredkin gates the base for achieving reversibility at circuit level, hence allowing to restore not only the state but also the energy, the fact that suitable membrane systems can simulate these circuits is quite relevant.

2 Background

We first fix some notation. With $\mathbb{N}$ we denote the set of *natural numbers* including zero, and with $\mathbb{N}^+$ the set of positive natural numbers. Given a set X, with $\mathbf{2}^X$ we indicate the set of subsets of X and with $\mathbf{2}^X_{\mathit{fin}}$ the set of *finite* subsets of X.

Given a set X, a partial order $\sqsubseteq$ on X is a reflexive, transitive and anti-symmetric relation. Let $(X, \sqsubseteq)$ be a partially ordered set and $Y \subseteq X$, we say that Y has a *minimum* iff there exists $x \in X$ such that $\forall y \in Y$ it holds that $x \sqsubseteq y$. Dually it has a *maximum* iff there exists $x \in X$ such that $\forall y \in Y$ it holds that $y \sqsubseteq x$. The elements of $Y \subseteq X$ are referred to as *incomparable* iff $\forall y, y' \in Y.\ y \neq y'$ implies that $y \not\sqsubseteq y'$ and $y' \not\sqsubseteq y$. Given a partial order $(X, \sqsubseteq)$, with $\mathit{max}(X, \sqsubseteq)$ we denote the set of elements $Y \subseteq X$ such that (a) for each $y \in Y$ and for each $x \in X$ if $y \sqsubseteq x$ then $y = x$ (the element y is not dominated by any other element of X), and (b) for each $x \in X$ such that there is no $x' \in X$ with $x' \neq x$ and $x \sqsubseteq x'$, then $x \in Y$ (the set is the greatest subset of incomparable and maximal elements of X), and similarly with $\mathit{min}(X, \sqsubseteq)$ we denote the greatest subset of elements $Y \subseteq X$ that are minimal with respect to the partial order relation. Given two elements $x, y \in X$ such that $x \sqsubseteq y$, we say that x is an *immediate* predecessor of y iff $x \neq y$ and $\forall z \in X.\ x \sqsubseteq z \sqsubseteq y$ either implies $x = z$ or $z = y$. If x is the immediate predecessor of y, we indicate this with $x \mathrel{\hat{\sqsubseteq}} y$.

A partial order $(X, \sqsubseteq)$ is a *tree* if $\sqsubseteq$ is such that each subset $Y \subseteq X$ of incomparable elements has no maximum, and each subset $Y \subseteq X$ has a minimum. The minimum of X is called the *root* of the tree. We define some auxiliary partial functions over trees. Given a tree $(X, \sqsubseteq)$, we define the partial function $\mathsf{father} : X \to X$ by $\mathsf{father}(x) = y$ whenever $y \mathrel{\hat{\sqsubseteq}} x$. Clearly, the root of a tree has no father. The function $\mathsf{children} : X \to \mathbf{2}^X$ is defined by $\mathsf{children}(x) = \{y \in X \mid x \mathrel{\hat{\sqsubseteq}} y\}$. If x is a leaf, then $\mathsf{children}(x) = \emptyset$. We assume that the trees have a finite degree, namely for each node x we assume that $\mathsf{children}(x) \in \mathbf{2}^X_{\mathit{fin}}$.

Multisets. Given a set S, a *multiset* over S is a function $m : S \to \mathbb{N}$; we denote by ∂S the set of multisets of S. The *multiplicity* of an element s in m is given by $m(s)$. A multiset m over S is *finite* iff the set $\mathsf{dom}(m) = \{s \in S \mid m(s) \neq 0\}$ is finite and we always consider finite multisets. A multiset m such that $\mathsf{dom}(m) = \emptyset$ is called *empty*, and it is denoted by $\mathbf{0}$. The cardinality of a multiset is defined as $\#(m) = \sum_{s \in S} m(s)$. Given a multiset in ∂S and a subset $S' \subseteq S$, by $m|_{S'}$

we denote the multiset over S' such that $m|_{S'}(s) = m(s)$. We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$, and $m \subset m'$ if $m \subseteq m'$ and $m \neq m'$. The operator $\oplus$ denotes *multiset union*: $(m \oplus m')(s) = m(s) + m'(s)$. The operator $\ominus$ denotes *multiset difference*: $(m \ominus m')(s) = \text{if } m(s) > m'(s) \text{ then } m(s) - m'(s) \text{ else } 0$. The *scalar product* of a number j with a multiset m is $(j \cdot m)(s) = j \cdot (m(s))$. We sometimes write a multiset $m \in \partial S$ as the sum $\oplus_{s \in S} m(s) \cdot s$, where we omit the summands whenever $m(s)$ is equal to 0. Finally we assume that all the operations defined so far extend (with overloading of notation) to vectors of multisets, applying the operations component-wise.

Membranes structure. The language of *membrane structure*, which we will denote with MS, is a language over the alphabet $\{[,]\}$, and it is defined inductively as follows:

- $[\,] \in \mathsf{MS}$, and
- if $\mu_1, \ldots, \mu_n \in \mathsf{MS}$ then also $[\mu_1 \ldots \mu_n] \in \mathsf{MS}$.

Two words in MS are *equivalent* whenever they represent the same tree up to isomorphisms, and a *membrane* μ is the equivalence class of all the words with respect to this equivalence. Observe that, given a membrane μ, a *matching pair* of parentheses is any substring of μ which is again a membrane. The number of membranes appearing in a membrane μ is calculated as follows:

$$\#_{\mathsf{MS}}(\mu) = \begin{cases} 1 & \text{if } \mu = [\,] \\ 1 + \sum_{i=1}^{k} \#_{\mathsf{MS}}(\mu_i) & \text{if } \mu = [\mu_1 \ldots \mu_k] \end{cases}$$

and to each membrane μ' appearing in a membrane μ, including μ itself, it is possible to associate an unique index i ranging from 1 to $\#_{\mathsf{MS}}(\mu)$, and we denote this index with $\mathsf{index}(\mu')$. If $\mu_i = [\mu_{i_1} \ldots \mu_{i_k}]$ then $\mathsf{father}(i_j) = i$ for $1 \leq j \leq k$, and $\mathsf{children}(i) = \{i_1, \ldots, i_k\}$. We assume that the index 1 is given to the root. Obviously the set $(\{1, \ldots, \#_{\mathsf{MS}}(\mu)\}, \sqsubseteq^*)$ is a tree, where $\mathsf{index}(\mu') \sqsubset \mathsf{index}(\mu_i)$ whenever $\mu' = [\mu_1 \ldots \mu_k]$, with $1 \leq i \leq k$, and $\sqsubseteq^*$ is the reflexive and transitive closure of $\sqsubset$.

3 Membrane Systems

We are now ready to recall the notion of *membrane system*. The main ingredients of a membrane system are three: a membrane structure, a multiset of objects associated to each membrane and a set of *evolution rules* associated to each membrane. The membrane structure represents the various compartments where the computations take place (in general simultaneously), and the conditions under which certain evolution rules can be applied is checked *locally*, *i.e.* in the same membrane to which the rules are associated. The result of the application of a rule has a more global effect, as it will be clear in the following.

We fix a finite alphabet of (names of) objects (sometimes called molecules), that we denote with $\mathcal{O}$ and we fix an alphabet of *rule names*, that will be denoted with Name, and it will be ranged over by $\mathfrak{n}$.

Definition 1. *A* membrane system *over a set of objects $\mathcal{O}$ is a construct $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ where:*

- *μ is a* membrane structure *with n membranes indexed from 1 to n, where $n = \#_{\mathsf{MS}}(\mu)$,*
- *each w_i^0 is a multiset over $\mathcal{O}$ associated with membrane i, and*
- *each R_i is a finite set of* reaction (or evolution) rules *r associated with the membrane i, each rule having the format $r : u \to v$, where u is a non empty finite multiset in $\partial\mathcal{O}$, v is a finite multiset over $\mathcal{O} \times (\{\mathsf{here}, \mathsf{out}\} \cup \{\mathsf{in}_j \mid \mathsf{father}(j) = i\})$, and $\mathsf{name}(r) \in \mathsf{Name}$ is the* name *of the rule r.*

The definition is almost standard, the difference is that we omitted the output membrane which is usually considered when one wishes to focus on what is calculated by a membrane system, and we focus on a rule format where a multiset of objects of a membrane are possibly *transformed* in multisets of objects in the same membrane and in the neighbouring ones (*i.e.* the father and the children). Two rules r, r' belonging to different sets of reaction rules (thus associated to different membranes) may be equal, where equal means that if $r : u \to v$ and $r' : u' \to v'$ then $u = u'$ and $v = v'$. We however assume that all the rules in a membrane system have distinct names, *i.e.* for each $r, r' \in \bigcup_{1 \leq i \leq n} R_i$, if $r \neq r'$ then $\mathsf{name}(r) \neq \mathsf{name}(r')$ and if $r = r'$ then there exists $k, j \in \{1, \ldots, n\}$ such that $r \in R_k$, $r' \in R_j$, $k \neq j$ and $\mathsf{name}(r) \neq \mathsf{name}(r')$. Given a rule $r \in \bigcup_{1 \leq i \leq n} R_i$, with $\mathsf{index}(r)$ we denote the index of the membrane this rule is associated to, thus if $r \in R_i$ then $\mathsf{index}(r) = i$.

The application of a rule $r : u \to v$ in a membrane i will *consume* the multiset u that must be in the membrane i and may cause the *production* of multisets not only in the same membrane i but also in the neighbouring membranes, if there are, namely those that are children of i and the $\mathsf{father}(i)$ membrane, if this exists. With $\mathcal{I}(r)$ we denote the set with the indices of the membranes where a rule r actually produces an object. Given a rule r, u is the *left hand side* of r and v is the *right hand side* of r, and they are denoted with $\mathsf{lhs}(r)$ and $\mathsf{rhs}(r)$, respectively. To simplify the notation, given a multiset z over $\mathcal{O} \times (\{\mathsf{here}, \mathsf{out}\} \cup \{\mathsf{in}_j \mid \mathsf{father}(j) = i\})$, with $z|_\alpha$ we denote the multiset on $\mathcal{O}$ obtained from z by considering all the elements with the second component equal to α, where $\alpha \in \{\mathsf{here}, \mathsf{out}, \mathsf{in}_1, \ldots, \mathsf{in}_n\}$. Given a rule r, its $\mathsf{rhs}(r) = v$ may be represented as $(v|_{\mathsf{here}}, \mathsf{here}) \oplus (v|_{\mathsf{out}}, \mathsf{out}) \oplus (v|_{\mathsf{in}_{j_1}}, \mathsf{in}_{j_1}) \oplus \cdots \oplus (v|_{\mathsf{in}_{j_k}}, \mathsf{in}_{j_k})$ where $\{j_1, \ldots, j_k\} = \mathsf{children}(\mathsf{index}(r))$. Observe that it may be that some of the $v|_\alpha$ are equal to $\mathbf{0}$. Given a rule r, the indices involved in the effect of this rule are $\{\mathsf{index}(r) \mid \mathsf{rhs}(r)|_{\mathsf{here}} \neq \mathbf{0}\} \cup \{\mathsf{father}(\mathsf{index}(r)) \mid \mathsf{rhs}(r)|_{\mathsf{out}} \neq \mathbf{0}\} \cup \{i \mid i \in \mathsf{children}(\mathsf{index}(r)) \ \wedge \ \mathsf{rhs}(r)|_{\mathsf{in}_i} \neq \mathbf{0}\}$. We assume that, for each rule r, it holds that $\mathcal{I}(r) \neq \emptyset$, hence each rule has an effect different from the *annihilation* of all the objects involved[2].

[2] This requirement is reasonable when one imagine that reversing means undoing the effects of a rule, thus if a rule just serves to annihilate all the objects to be rewritten then one can imagine that such a rule can be always reversed, in any multiplicity.

Membrane Systems Evolution. A membrane system Π evolves from a configuration to another configuration as a consequence of the application of (multisets of) rules in each region. The rules are applied *simultaneously.* We start formalizing the notion of configuration of a membrane system.

Definition 2. *Let* $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ *be a membrane system, then a* configuration *is a tuple* $C = (w_1, \ldots, w_n)$ *where each* w_i *is a multiset over* $\mathcal{O}$. $C_0 = (w_1^0, \ldots, w_n^0)$ *is the* initial *configuration of* Π.

A computation step of a membrane system is *triggered* by the application of multisets of rules in each membrane. These multisets of rules are collected in a vector.

Definition 3. *Let* $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ *be a membrane system, then a* multi-rule vector $\overrightarrow{R}$ *is the tuple* $(\widehat{R}_1, \ldots, \widehat{R}_n)$, *where* $\widehat{R}_i$ *is a multiset over* R_i.

The multi-rule vector $\overrightarrow{R}$ contains all the rules that have to be applied simultaneously to a configuration of a membrane system, with their proper multiplicities.

A multi-rule vector $\overrightarrow{R}$ is *enabled* at a configuration C whenever each multiset of objects in each region is greater than or equal to what all the rules to be applied in that region consume. Given a multi-rule vector $\overrightarrow{R}$, for each i between 1 and n we denote with $\mathsf{Lhs}(\overrightarrow{R})_i$ the multiset over $\mathcal{O}$ defined as follows: $\bigoplus_{r \in R_i} \widehat{R}_i(r) \cdot \mathsf{lhs}(r)$. The tuple of these multisets is denoted with $\mathsf{Lhs}(\overrightarrow{R})$.

Definition 4. *Let* $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ *be a membrane system,* $\overrightarrow{R}$ *a multi-rule vector and* C *a configuration. Then* $\overrightarrow{R}$ *is* enabled *at* $C = (w_1, \ldots, w_n)$ *if* $\forall i \in \{1, \ldots, n\}$. $\mathsf{Lhs}(\overrightarrow{R})_i \subseteq w_i$. *We denote the enabling of a multi-rule vector* $\overrightarrow{R}$ *at a configuration* C *with* $C\,[\overrightarrow{R}\rangle$.

The effects of the *application* of a multi-rule vector $\overrightarrow{R}$ (which acts in all membranes concurrently) in the membrane i are the following: the multiset of objects $\bigoplus_{r \in R_i} \widehat{R}_i(r) \cdot \mathsf{rhs}(r)|_{\mathsf{here}}$ is the effect of the rules in the same membrane, $(\bigoplus_{r \in R_{\mathsf{father}(i)}} \widehat{R}_{\mathsf{father}(i)}(r) \cdot \mathsf{rhs}(r)|_{\mathsf{in}_i})$ those of the rules in the father membrane, and finally $(\bigoplus_{j \in \mathsf{children}(i)} (\bigoplus_{r \in R_j} \widehat{R}_i(r) \cdot \mathsf{rhs}(r)|_{\mathsf{out}}))$ those from the children membranes. Like previously, these three parts are combined by using $\oplus$. For each membrane, we denote the effects by $\mathsf{Rhs}(\overrightarrow{R})_i$. The tuple of these effects is written as $\mathsf{Rhs}(\overrightarrow{R})$.

The following definition captures the notion of evolution of a membrane system with the application of a multi-rule vector $\overrightarrow{R}$.

Definition 5. *Let* $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ *be a membrane system,* $C = (w_1, \ldots, w_n)$ *be a configuration and* $\overrightarrow{R} = (\widehat{R}_1, \ldots, \widehat{R}_n)$ *be a multi-rule vector such that* $C\,[\overrightarrow{R}\rangle$. *Then* $\overrightarrow{R}$ *can be* executed *and its execution leads to a configuration* $C' = (w'_1, \ldots, w'_n)$ *where* $w'_i = w_i \ominus \mathsf{Lhs}(\overrightarrow{R})_i \oplus \mathsf{Rhs}(\overrightarrow{R})_i$. *The execution of a multi-rule vector* $\overrightarrow{R}$ *at a configuration* C *is denoted with* $C\,[\overrightarrow{R}\rangle\, C'$.

For the enabling and the execution of a multi-rule vector we adopt a notation resembling the one usually adopted for Petri nets, also because of the tight connections among these two formalisms (see [6,7,12,13,20] among others, or the chapter in [23]). Sometimes we will call an evolution step of a membrane system as a *reaction step*.

We now formalize the *chain* of "reactions" for a given membrane system: C_0 is a reaction sequence, and if $C_0\,[\overrightarrow{R}_1\rangle\, C_1 \ldots C_{n-1}\,[\overrightarrow{R}_n\rangle\, C_n$ is a reaction sequence, and $C_n\,[\overrightarrow{R}\rangle\, C$, then $C_0\,[\overrightarrow{R}_1\rangle\, C_1 \ldots C_n\,[\overrightarrow{R}\rangle\, C$ is also a reaction sequence. A configuration C is said to be *reachable* if there is a reaction sequence starting from the initial configuration and leading to C, *i.e.* $C_0\,[\overrightarrow{R}_1\rangle\, C_1 \ldots C_{n-1}\,[\overrightarrow{R}_n\rangle\, C_n$ with $C = C_n$.

The evolution of membrane systems may have several strategies, and usually it is assumed that in each membrane all the applicable rules are actually applied in a *maximally parallel* way. Thus if $\overrightarrow{R}$ is enabled at the configuration C ($C\,[\overrightarrow{R}\rangle$) it is implicitly assumed that there is no rule r in any of the rules sets R_i such that $C\,[\overrightarrow{R}'\rangle$ where $\overrightarrow{R'}$ is obtained from $\overrightarrow{R}$ adding an instance of the rule r to the proper multiset. However, other strategies may be used, for instance maximality with respect to a specific membrane index (no rule associated to that membrane can be added to the multi-rule vector), or the rules to be applied are those involving the presence of a specific object called catalyst, or to each rule a readiness index can be associated and the criteria is to maximize the sum of these indices, or simply a priority can be attached to each rule and those enabled with highest priorities have to be applied. The various strategies that can be adopted have an influence on the expressiveness of the paradigm, that is not our concern, as we already mentioned in the introduction.

4 Reversing Membrane System Computations

Reversibility in membrane systems is strongly connected to the idea that computations are *deterministic*. Here we consider an approach which is more similar to the one taken when reversibility is considered in the realm of distributed and concurrent computations.

Rather than introducing new rules (reversed, like in dual membrane systems where the effect of *undoing* is obtained applying reversed rules) we formalize what the reverse application of a multi-rule vector is.

Definition 6. *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $C = (w_1, \ldots, w_n)$ a configuration and $\overrightarrow{R}$ be a multi-rule vector. Then $\overrightarrow{R}$ is* reversely enabled *at C whenever, for all $i \in \{1, \ldots, n\}$, it holds that $\mathsf{Rhs}(\overrightarrow{R})_i \subseteq w_i$, and it is denoted with $C\,\langle\overrightarrow{R}]$.*

The intuition is almost trivial: the enabling is done by checking on the effects of the application of rules. Observe that this fits easily when rule formats like symport/antiport are considered, or like in [7] where a more general format for rules is considered.

Definition 7. *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $C = (w_1, \ldots, w_n)$ a configuration and $\overrightarrow{R}$ be a multi-rule vector such that $C \,\langle \overrightarrow{R}]$. Then $\overrightarrow{R}$ can be* reversed *and the effects of reversing this multi-rule vector are, for all $i \in \{1, \ldots, n\}$, $w_i' = w_i \ominus \mathsf{Rhs}(\overrightarrow{R})_i \oplus \mathsf{Lhs}(\overrightarrow{R})_i$. We write $C \,\langle \overrightarrow{R}]\, C'$ to state that the configuration C' is the effect of reversing the multi-rule vector $\overrightarrow{R}$. In this case we say that $\overrightarrow{R}$ is* reversely executed.

Once we have established what reversely enabling and reverse execution might be, we start to connect these notion with the usual *forward* executions.

Proposition 1. *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $C = (w_1, \ldots, w_n)$ a configuration and $\overrightarrow{R}$ be a multi-rule vector such that $C\,[\overrightarrow{R}\rangle$, and let C' be the configuration reached executing $\overrightarrow{R}$,* i.e. *$C\,[\overrightarrow{R}\rangle\, C'$. Then $C' \,\langle \overrightarrow{R}]$.*

The loop lemma can be easily proven also in this setting:

Lemma 1 (Loop lemma). *Let $\Pi = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $C = (w_1, \ldots, w_n)$ a configuration and $\overrightarrow{R}$ be a multi-rule vector such that $C\,[\overrightarrow{R}\rangle$, and let C' be the configuration reached executing $\overrightarrow{R}$,* i.e. *$C\,[\overrightarrow{R}\rangle\, C'$. Then $C' \,\langle \overrightarrow{R}]\, C$.*

The proof of the following theorem is obvious.

Theorem 1. *Let Π be a membrane system, C a configuration and $\overrightarrow{R}$ be a multi-rule vector such that $C\,[\overrightarrow{R}\rangle C'$. Then there exists a multi-rule vector $\overrightarrow{R'}$ such that $C' \,\langle \overrightarrow{R'}]\, C$.*

Observe that not necessarily $\overrightarrow{R'}$ should be equal to $\overrightarrow{R}$. In fact they may differ.

Example 1. Consider the membrane system with just one membrane, the unique rule associated to the membrane are $r^1 = a \rightarrow (b, \mathsf{here})$ and $r^2 = a \oplus b \rightarrow (2b, \mathsf{here})$, and the initial configuration is $a \oplus b$. The rule r^1 is enabled at the initial configuration and its application leads to the configuration $2b$. Now also r^2 can be reversely applied at this configuration and the initial configuration can be obtained again.

The main problem is that membrane systems do not keep any information about the past, thus at a certain configuration it could be that a multi-rule vector $\overrightarrow{R}$ can be reversely executed even when no $\overrightarrow{R}'$ such that $\overrightarrow{R} \subseteq \overrightarrow{R}'$ has been "forwardly" executed. This contrasts the idea that reversibility is like undoing something that has been done previously.

Example 2. Consider the membrane system with 2 membranes $[\ [\ ]_2\ ]_1$, where the indices are the ones associated to the membranes, and with the following sets of rules: $\{r_1^1 : 2a \rightarrow (a \oplus b, \mathsf{here}) \oplus (b, \mathsf{in}_2), r_2^1 : b \rightarrow (a, \mathsf{here}) \oplus (c, \mathsf{in}_2), r_3^1 : a \oplus b \rightarrow$

$(2a, \mathsf{here}) \oplus (b, \mathsf{in}_2), r_4^1 : a \to (b, \mathsf{here}) \oplus (c, \mathsf{in}_2), r_5^1 : 2a \to (b \oplus c, \mathsf{in}_2)\}$ are the rules associated to the first membrane, and $\{r_1^2 : b \to (2a, \mathsf{out}), r_2^2 : c \to (b, \mathsf{out}), r_3^2 : b \to (a, \mathsf{out}), r_4^2 : c \to (c, \mathsf{out})\}$ are those associated to the second membrane. The initial configuration is (w_1^0, w_2^0) where $w_1^0 = 2a \oplus b$ and $w_2^0 = \mathbf{0}$. The configuration $(2a \oplus b, b \oplus c)$ can be reached either executing the multi-rule vector $(r_1^1 \oplus r_2^1, \mathbf{0})$ or the one $(r_3^1 \oplus r_4^1, \mathbf{0})$. At this configuration these two multi-rule vectors are reversely enabled, but also the multi-rule vector $(r_5^1, \mathbf{0})$, and reversely executing it we would obtain the configuration $(2a, \mathbf{0})$ which is not reachable using the rules in the membrane system.

A similar problem is present in all the algebraic process calculi for which reversibility has been studied (see [8,10,14,15] among others). The solution is usually to add a *memory* which helps to keep track of the evolution of the processes. Here we pursue a similar idea by adding information to configurations (membranes). We assume that Name contains $\perp$ as a name which is not associated to any rule.

4.1 Membranes with Memory

Objects of a membrane system may be enriched by adding the name of the rule producing them. Thus objects would be $\mathcal{O} \times \mathsf{Name}$, and reversing a step would be to find out whether there are enough objects with specific rules names. The *forward* enabling would ignore the information on which rule produced the object, and the execution of the step would simply add the proper name of each object produced. This solution allow to undo just one step, as the information on the name of the rule of the *consumed* object are lost.

To be able to undo more steps we have to figure out a different structure, which we call memory and we will add it to configurations.

We briefly discuss what the *memory* in this case could be. The idea is rather simple: the memory is a labeled partial order, where the labeling gives a triple composed by an object, the index of a membrane and a rules name, thus $\langle o, i, \mathfrak{n} \rangle$ conveys the idea that the object o has been produced in the membrane i using the rule $\mathfrak{n}$.

Definition 8. *Let* Name *be a set of rules names such that* $\perp \in \mathsf{Name}$, *let* $\mathcal{O}$ *be a set of objects and let* $\mathcal{I}$ *be a set of indices. Then a* memory m *is the labeled partial order* $(X, \preceq, l)$ *where* $(X, \preceq)$ *is a partial order and* $l\colon X \to \mathcal{O} \times \mathcal{I} \times \mathsf{Name}$ *is a labeling mapping With* Mem *we denote the set of memories.*

Given an element of $(o, i, \mathfrak{n}) \in \mathcal{O} \times \mathcal{I} \times \mathsf{Name}$, we define some obvious projections operators, that carry over on multistes of $\mathcal{O} \times \mathcal{I} \times \mathsf{Name}$. $\mathsf{obj}_m\colon \mathcal{O} \times \mathcal{I} \times \mathsf{Name} \to \mathcal{O}$ is defined as $\mathsf{obj}_m(o, i, \mathfrak{n}) = o$, $\mathsf{i}_m\colon \mathcal{O} \times \mathcal{I} \times \mathsf{Name} \to \mathcal{I}$ as $\mathsf{i}_m(o, i, \mathfrak{n}) = i$, and finally $\mathsf{rule}_m\colon \mathcal{O} \times \mathcal{I} \times \mathsf{Name} \to \mathsf{Name}$ as $\mathsf{rule}_m(o, i, \mathfrak{n}) = \mathfrak{n}$. Given $\mathsf{m} = (X, \preceq, l)$, with $\mathsf{max}(\mathsf{m})$ we denote the (multi)set $\oplus_{x \in max(X, \preceq)} l(x)$.

On memories we define two operations: one to add a vertex and another one to remove a vertex. These operations are obviously extended to sets of vertices. Given an element $\mathsf{a} \in \mathcal{O} \times \mathcal{I} \times \mathsf{Name}$ and a set of vertices $Y \subseteq X$, with add we denote the operation that takes a memory $\mathsf{m} = (X, \preceq, l)$, the set of vertices Y and the element a and add a new vertex, labeled with a, which is greater than all the vertex in Y. Formally $\mathsf{add}(\mathsf{m}, Y, \mathsf{a})$ is the memory $\mathsf{m}' = (X \cup \{y\}, \preceq', l')$ where $y \notin X$, $l'(y) = \mathsf{a}$ and $l'(x) = l(x)$ if $x \in X$, and $\preceq'$ is obtained closing transitively and reflexively the relation $\preceq \ \cup \{(y', y) \mid y' \in Y\}$ (though not explicitly stated here, we imagine that the set Y is not empty and is a subset of $max(X, \preceq)$). With remove we denote the operation of removing a vertex x from a memory, thus given a memory $\mathsf{m} = (X, \preceq, l)$, and $x \in X$, with $\mathsf{remove}(\mathsf{m}, x)$ we denote the memory $\mathsf{m}' = (X \backslash \{x\}, \preceq', l')$ where $\preceq'$ and l' are the restriction of $\preceq$ and l respectively to $X \backslash \{x\}$ (though not explicitly stated here, we imagine that only maximal elements are removed). We do need some further notation. Consider a multiset z over $\mathcal{O} \times \{1, \ldots, n\} \times \mathsf{Name}$, and an index $i \in \{1, \ldots, n\}$, with $\lfloor z \rfloor_i$ we denote the multiset defined as follows: $\lfloor z \rfloor_i(a) = z(a)$ if $\mathsf{i}_m(a) = i$ and $\lfloor z \rfloor_i(a) = 0$ otherwise.

The notion of membrane system does not change, it changes however the one of configuration (than now has a memory).

Definition 9. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system. Then a configuration with memory is the pair $\mathcal{C} = (C, \mathsf{m})$ where $C = (w_1, \ldots, w_n)$ is the tuple of multisets over $\mathcal{O}$ and $\mathsf{m} = (X, \preceq, l)$ is a memory such that for each $i \in \{1, \ldots, n\}$ it holds that $w_i = \mathsf{obj}_m(\lfloor \mathsf{max}(\mathsf{m}) \rfloor_i)$.*

The initial configuration $\mathcal{C}_0$ is the pair (C_0, m_0), where $C_0 = (w_1^0, \ldots, w_n^0)$ and $\mathsf{m}_0 = (X, \preceq, l)$ is a memory such that $\forall x \in X$, $\mathsf{rule}_m(l(x)) = \bot$ and $\forall x, y \in X$. $x \preceq y$ implies $x = y$.

Given a configuration with memory $\mathcal{C} = (C, \mathsf{m})$, then $\eta(\mathcal{C})$ is C and $\gamma(\mathcal{C})$ is m.

A configuration has now a memory and the requirement is that for each maximal element of the memory corresponds an object in the membrane configuration. The initial memory is such that the maximal elements carry the information on the rule stating that they have not been produced by any rule, and the partial ordering is the discrete one.

Example 3. Consider the membrane system with just one membrane with the set of rules: $\{r_1^1 : a \to (a, \mathsf{here}), r_2^1 : a \to (2a, \mathsf{here}), r_3^1 : b \to (a \oplus b, \mathsf{here}), r_4^1 : a \oplus b \to (a, \mathsf{here})\}$ and the following initial configuration: $(a \oplus b, (\{v_1, v_2\}, id, l))$, where id is the identity relation on $\{v_1, v_2\}$, $l(v_1) = (a, 1, \bot)$ and $l(v_2) = (b, 1, \bot)$.

The definition of enabling of a multi-rule vector is the same as for membrane systems: it should be checked on the object part of a configuration (which is closely related to the memory).

Definition 10. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C} = (C, \mathsf{m})$ a configuration with memory, and $\overrightarrow{R}$ a multi-rule vector. Then $\overrightarrow{R}$ is* enabled *at $\mathcal{C}$ whenever $\eta(\mathcal{C})\,[\overrightarrow{R}\rangle$, We denote the enabling of $\overrightarrow{R}$ at $\mathcal{C}$ with $\mathcal{C}\,\{\![\overrightarrow{R}\rangle$.*

Consider a memory $\mathsf{m} = (X, \preceq, l)$ and a subset of vertex $Y \subseteq max(X, \preceq)$ with $\widetilde{\mathsf{max}}(Y)$ we denote the multiset $\oplus_{y \in Y} l(y)$.

Given a configuration with memory $\mathcal{C} = ((w_1, \dots, w_n), \mathsf{m})$ and a multi-rule vector $\overrightarrow{R}$, for each rule r such that $\widehat{R}_{\mathsf{index}(r)}(r) > 0$, with $\mathsf{LHS}_m(r)$ we denote the pair $(u_{\mathsf{index}(r)}, Y)$ where $u_{\mathsf{index}(r)} \subseteq w_{\mathsf{index}(r)}$ is such that $\mathsf{lhs}(r) = u_{\mathsf{index}(r)}$, with $w_{\mathsf{index}(r)}$ in $\eta(\mathcal{C})$, and Y is a subset of the maximal elements in $\gamma(\mathcal{C}) = (X, \preceq, l)$ such that $\lfloor \widetilde{\mathsf{max}}(Y) \rfloor_{\mathsf{index}(r)} = u_{\mathsf{index}(r)}$.

Once a multi-rule vector $\overrightarrow{R}$ is enabled at a configuration with memory we have to state the effects of the application of a rule r. The idea is now the following: for each object of the multiset *produced* by a rule we add to the memory a new vertex labeled with the object, the membrane index it belongs to, and the name of rule r.

Consider a rule r enabled at a configuration $\mathcal{C}$, and consider $\mathsf{LHS}_m(r) = (u_{\mathsf{index}(r)}, \{Y\})$. Consider now $\mathsf{rhs}(r)$, and take $\mathsf{rhs}(r)|_\alpha$ with $\alpha \in \{\mathsf{here}, \mathsf{out}\} \cup \{\mathsf{in}_i \mid \mathsf{father}(i) = \mathsf{index}(r)\}$. Then $\mathsf{RHS}_m(r)_i$ is the multiset in $\mathcal{O}$ defined as usual as $\mathsf{Rhs}(r)_i$, and the new memory is obtained from $\gamma(\mathcal{C}) = (X, \preceq, l)$ by adding for each object o in $\mathsf{RHS}_m(r)_i$ a new vertex y greater than any vertex in Y and labeled with $(o, i, \mathsf{name}(r))$. We denote this operation as $\mathsf{Add}(\gamma(\mathcal{C}), \mathsf{RHS}_m(r)_i, Y)$ and it is the extension of the operation add defined previously. Given a multi-rule vector $\overrightarrow{R}$, for each i between 1 and n, with overloading of notation, we denote with $\mathsf{LHS}_m(\overrightarrow{R})_i$ the multiset of pairs over $\mathcal{O}$ and set of subsets of indices, defined as $\bigoplus_{r \in R_i} \widehat{R}_i(r) \cdot \mathsf{LHS}_m(r)$ (where the sum for pairs acts as the sum on the multiset part and union on the other), and the tuple of these pairs is denoted with $\mathsf{LHS}_m(\overrightarrow{R})$, $\widehat{\mathsf{LHS}_m}(\overrightarrow{R})$ is the tuple obtained considering only the first components of $\mathsf{LHS}_m(\overrightarrow{R})$ (thus $\mathsf{Lhs}(\overrightarrow{R})$), and $\overbrace{\mathsf{LHS}_m}(\overrightarrow{R})$ is the set of subsets of vertices and it is such that $\forall Y, Y' \in \overbrace{\mathsf{LHS}_m}(\overrightarrow{R})$, $Y \neq Y'$ implies that $Y\mathcal{Y}' = \emptyset$ (all the involved vertices are distinct). Similarly, for each membrane, we denote the effects by $\mathsf{RHS}_m(\overrightarrow{R})_i$ and $\mathsf{RHS}_m(\overrightarrow{R})$ denotes the tuple of these effects and on memory is $\mathsf{Add}(\gamma(\mathcal{C}), \mathsf{RHS}_m(\overrightarrow{R}), \overbrace{\mathsf{LHS}_m}(\overrightarrow{R}))$ where $\overbrace{\mathsf{LHS}_m}(\overrightarrow{R})$ is a *set* of subset of the maximal elements in $\gamma(\mathcal{C})$ that have to be followed by the new objects (thus there is a set of maximal elements for each applied rule).

Definition 11. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \dots, w_n^0, R_1, \dots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration with memory, and $\overrightarrow{R}$ a multi-rule vector such that $\mathcal{C}\,\{\![\overrightarrow{R}\rangle$, and assume that $\overbrace{\mathsf{LHS}_m}(\overrightarrow{R})$ is the list of maximal elements of $\gamma(\mathcal{C})$ as described above. Then $\mathcal{C}\,\{\![\overrightarrow{R}\rangle\mathcal{C}'$ where $\mathcal{C}'$ is obtained by $\mathcal{C}$ as follows: for each membrane index i, $w_i' = w_i \ominus \mathsf{Lhs}(\overrightarrow{R})_i \oplus \mathsf{Rhs}(\overrightarrow{R})_i$ and the memory is $\mathsf{Add}(\gamma(\mathcal{C}), \mathsf{RHS}_m(\overrightarrow{R}), \overbrace{\mathsf{LHS}_m}(\overrightarrow{R}))$.*

The definition is rather obvious: for each object o produced in a membrane i by the rule $\mathfrak{n}$ a new vertex is added in the memory which is greater than the elements *consumed* by the rule.

Observe that the elements added to the configuration are precisely among the maximal elements in the memory.

Proposition 2. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration with memory, and $\overrightarrow{R}$ a multi-rule vector such that $\mathcal{C}\,\{\![\overrightarrow{R}\rangle\mathcal{C}'$. Take $Y = \mathsf{max}(\gamma(\mathcal{C}'))$ and consider $l(Y)$ which can be seen as a multiset over $\mathcal{O} \times \{1, \ldots, n\} \times \mathsf{Name}$. Then for each $i \in \{1, \ldots, n\}$. $\mathsf{obj}_m(\lfloor l(Y) \rfloor_i) = w'_i$ where $\eta(\mathcal{C}') = (w'_1, \ldots, w'_n)$.*

Example 4. Consider the membrane system of Example 3. At the initial configuration the following sets of rules are enabled: $\{r_1^1 \oplus r_3^1\}$, $\{r_2^1 \oplus r_3^1\}$, $\{r_4^1\}$. Consider the last one. The execution of it gives the configuration $((a, r_4^1), (\{v_1, v_2, v_3\}, id \cup \{(v_1, v_3), (v_2, v_3)\}, l'))$ where $l'(v_1) = l(v_1), l'(v_2) = l(v_2)$ and $l'(v_3) = (b, 1, r_4^1)$.

Performing another one, for instance $\{r_1^1 \oplus r_3^1\}$, would give a different memory.

We show that this is a conservative extension of membrane systems, as to each step in a membrane system with memory, a step corresponds in the membrane system where all the added information is forgotten.

Proposition 3. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system, $\mathcal{C}$ a configuration with memory, $\overrightarrow{R}$ a multi-rule vector such that $\mathcal{C}\,\{\overrightarrow{R}\rangle$, and let $\mathcal{C}\,\{\![\overrightarrow{R}\rangle\mathcal{C}'$. Then $\eta(\mathcal{C})\,[\overrightarrow{R}\rangle$ and $\eta(\mathcal{C})\,[\overrightarrow{R}\rangle\,\eta(\mathcal{C}')$.*

We discuss now when a rule r can be *reversely* applied in this setting. Again the intuition is rather simple, just check if there are enough objects bearing the name of the rule r among the maximal elements of the memory. Let m be a memory, $\mathfrak{n}$ be a rule name, and i a membrane index, then with $\blacktriangleleft^i_{\mathsf{name}(r)}(\mathsf{m})$ we denote the multiset on $\mathcal{O}$ defined as

$$\blacktriangleleft^i_{\mathsf{name}(r)}(\mathsf{m}) = \bigoplus_{x \in \mathsf{max}(\mathsf{m})} \{\mathsf{obj}_m(l(x)) \mid \mathsf{rule}_m(l(x)) = \mathsf{name}(r) \ \wedge \ \mathsf{i}_m(l(x)) = i\}$$

Let r be a rule and $\mathcal{C}$ be a configuration of a membrane system with memory Π_m. Then r is reversely enabled at $\mathcal{C} = ((w_1, \ldots, w_n), \mathsf{m})$ whenever, for all $k \in \mathcal{I}(r)$, $\mathsf{rhs}(r)_k \subseteq \blacktriangleleft^k_{\mathsf{name}(r)}(\mathsf{m})$. The reverse enabling is summarized in the following definition.

Definition 12. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration, and $\overrightarrow{R}$ a multi-rule vector. Then $\overrightarrow{R}$ is* reversely enabled *at $\mathcal{C}$ if for rule r in $\overrightarrow{R}$ $\widehat{R}_{\mathsf{index}(r)} \cdot \mathsf{rhs}(r)_k \subseteq \blacktriangleleft^{w_k}_{\mathsf{name}(r)}(\gamma(\mathcal{C}))$. The reverse enabling of a multi-rule vector is denoted with $C\,\langle\overrightarrow{R}]\!\}$.*

In this case we have to find, for each each instance of a given rule, enough objects produced by an instance of the same rule at the same (local) configuration.

Once a multi-rule vector is reversely enabled, it may be applied. We start showing what it means to undo a single rule r. Given a configuration $\mathcal{C}$, with the memory $\gamma(\mathcal{C}) = (X, \preceq, l)$, for each index $k \in \mathcal{I}(r)$ we have that $\mathsf{rhs}(r)_k$

is contained in $\blacktriangleleft^k_{\mathsf{name}(r)}$ $(\gamma(\mathcal{C}))$. Consider a subset $Y \subseteq max(X, \preceq)$ such that $\mathsf{obj}_m(\lfloor \oplus_{y\in Y} l(y)\rfloor_k) = \mathsf{rhs}(r)_k$, then what we have to do on the memory is just to remove the set Y from the memory. The set of these vertices are denoted with $\widehat{\mathsf{RHS}_m}(r)$ and it extends obviously to $\overrightarrow{R}$. Clearly we require that these sets of vertices are disjoint.

Definition 13. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C} = ((w_1, \ldots, w_n), \mathsf{m})$ be a configuration, and $\overrightarrow{R}$ a multi-rule vector such that $C \,\langle \overrightarrow{R}]\!\}$. Then $\mathcal{C}' = ((w_1', \ldots, w_n'), \mathsf{m}')$, where $w_i' = w_i \ominus \mathsf{Rhs}(\overrightarrow{R})_i \oplus \mathsf{Lhs}(\overrightarrow{R})_i$ and m' is obtained from m by removing all the vertex in m corresponding to the object in $\mathsf{RHS}_m(\overrightarrow{R})$, thus $\mathsf{m}' = \mathsf{remove}(\mathsf{m}, \widehat{\mathsf{RHS}_m}(\overrightarrow{R}))$, is the configuration reached by reversely executing $\overrightarrow{R}$ at $\mathcal{C}$. As before it is denoted with $C \,\langle \overrightarrow{R}]\!\}\, C'$.*

Example 5. Consider the membrane system of Example 3 and the computation step done in Example 4. The set $\{r_4^1\}$ is reversely enabled and

$$(b, (\{v_1, v_2, v_3\}, id \cup \{(v_1, v_3), (v_2, v_3)\}, l')) \ \langle \{r_4^1\}]\!\} \ (a \oplus b, (\{v_1, v_2\}, id, l))$$

where the labeling are those in Examples 3 and 4.

Consider another membrane system with just one membrane with the set of rules: $\{r_1^1 : b \to (a \oplus b, \mathsf{here})\}$ and the initial configuration (b, m_0). Applying to this configuration $\{r_1^1\}$ we have

$$(b, \mathsf{m}_0) \,\{\![\{r_1^1\} \rangle\, (a \oplus b, \mathsf{m}_1)$$

where $\mathsf{m}_1 = (\{v_1, v_2, v_3\}, \preceq, l)$ where $v_1 \preceq v_2, v_1 \preceq v_3$ and l is the following: $l(v_1) = (b, 1, \bot), l(v_2) = (a, 1, r_1^1)$ and $l(v_3) = (b, 1, r_1^1)$. To this configuration we can apply again the same rule:

$$(a \oplus b, \mathsf{m}_1) \ \ \{\![\{r_1^1\} \rangle \ \ (a \oplus a \oplus b, \mathsf{m}_2)$$

where now m_2 is $(\{v_1, v_2, v_3, v_4, v_5\}, \preceq', l')$ with $v_3 \preceq' v_4, v_3 \preceq' v_5$ and the new vertices are labelled as $l(v_4) = (a, 1, r_1^1)$ and $l(v_5) = (b, 1, r_1^1)$. Reversely applying the unique rule we could have now a choice: either consider the vertices $\{v_4, v_5\}$ or $\{v_2, v_5\}$. In the latter case we have

$$((a \oplus a \oplus b, \mathsf{m}_2) \ \ \langle \{r_1^1\} \} \, (a \oplus b, \mathsf{m}_3)$$

where m_3 is obtained from m_2 by removing the vertices v_2 and v_5. This choice (which is investigated in a different setting in [19]) has as consequence that we cannot further undo going back to the initial configuration.

If the vertices $\{v_4, v_5\}$ are taken into accont, then the configuration $(a \oplus b, \mathsf{m}_1)$ is obtained again.

Again the loop lemma can be proved also in this setting but, as the previous example points out, it is a weaker version with respect to the one we introduced previously.

Lemma 2 (Loop lemma for membrane system with memory). *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration, and $\overrightarrow{R}$ be a multi-rule vector such that $\mathcal{C}\,\{\![\overrightarrow{R}\rangle$, and let $\mathcal{C}'$ be the configuration reached by executing $\overrightarrow{R}$,* i.e. *$\mathcal{C}\,\{\![\overrightarrow{R}\rangle\,\mathcal{C}'$. Then there exists a set of vertices in $\gamma(\mathcal{C}')$ associated to the object to be removed by the reverse application of $\overrightarrow{R}$, such that $\mathcal{C}'\,\langle\overrightarrow{R}]\!\}\mathcal{C}''$ and $\mathcal{C} = \mathcal{C}''$.*

Observe that not necessarily the objects consumed by the application of a multi-rule vector are those used in the reverse application of it. Hence we not necessarily obtain again the same memory. However, if the memory is the same, then the vector multi-rule reversely applied is the same we started with.

Theorem 2. *Let Π_m be a membrane system with memory, $\mathcal{C}$ a configuration, and $\overrightarrow{R}$ be a multi-rule vector such that $\mathcal{C}\,\{\![\overrightarrow{R}\rangle\,\mathcal{C}'$. Then for all multi-rule vector $\overrightarrow{R'}$ such that $\mathcal{C}'\,\langle\overrightarrow{R'}]\!\}\,\mathcal{C}$ it holds that $\overrightarrow{R'} = \overrightarrow{R}$.*

Obviously, the reversing in a membrane system with memory and the reversing in the membrane system where the additional information are forgotten, are related in a precise way.

Proposition 4. *Let $\Pi_m = (\mathcal{O}, \mu, w_1^0, \ldots, w_n^0, R_1, \ldots, R_n)$ be a membrane system with memory, $\mathcal{C}$ a configuration, and $\overrightarrow{R}$ be a multi-rule vector such that $\mathcal{C}\,\{\![\overrightarrow{R}\rangle$, and let $\mathcal{C}'\,\langle\overrightarrow{R}]\!\}\,\mathcal{C}$. Then $\eta(\mathcal{C}')\,\langle\overrightarrow{R}]\,\eta(\mathcal{C})$.*

5 Future Works

Reversibility in membrane systems has several facets. One is connected with determinism and the fact that each configuration has just a single predecessor, another is related to the amount of information needed to *reconstruct* past configurations. Concerning this view of reversibility, we have proposed a way to add all the relevant informations to *undo* steps properly. It must be said that many other solutions are conceivable, depending on the amount of information needed, for instance objects may be enriched to carry the history. The approach we presented here has the characteristic that the memory not only allow to reverse steps properly but also keep tracks of the dependencies among steps and objects.

Beside continuing to investigate on how reversibility can be achieved in membrane systems, we put two possible research issues. Here we have considered that all the rules are reversible, but this assumption is a maybe too strong when computations that are inspired by nature are considered. We may imagine that some rules produce *irreversible* effects, that cannot be undone. This may be modelled simply forgetting the rules names in both approaches. However this opens many questions on how to actually reverse computations and also on the notions of causality as investigated in [20] or [2]. Various situations may be devised in this setting, similarly to what is done in [19]. Here some events are undone but still some of their effects may remain. This idea can be possibly implemented also in membrane systems, opening new interesting feature.

Another issue is the possibility of combining the two ways: a part of the multi-rule vector is used to compute forward, another part is used to undo some effects. Again this has to be fully investigated.

Acknowledgement. The author acknowledge the useful remarks and suggestions by the anonymous reviewers.

References

1. Agrigoroaiei, O., Ciobanu, G.: Reversing computation in membrane systems. J. Logic Algebraic Program. **79**(3–5), 278–288 (2010)
2. Agrigoroaiei, O., Ciobanu, G.: Rule-based and object-based event structures for membrane systems. J. Logic Algebraic Program. **79**(6), 295–303 (2010)
3. Alhazov, A., Freund, R., Morita, K.: Sequential and maximally parallel multiset rewriting: reversibility and determinism. Nat. Comput. **11**(1), 95–106 (2012)
4. Aman, B., Ciobanu, G.: Computational power of protein interaction networks. In: Mauri, G., Dennunzio, A., Manzoni, L., Porreca, A.E. (eds.) UCNC 2013. LNCS, vol. 7956, pp. 248–249. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39074-6_25
5. Ciobanu, G., Pan, L., Păun, G., Pérez-Jiménez, M.J.: P systems with minimal parallelism. Theoret. Comput. Sci. **378**(1), 117–130 (2007)
6. Ciobanu, G., Pinna, G.M.: Catalytic Petri nets are turing complete. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 192–203. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28332-1_17
7. Ciobanu, G., Pinna, G.M.: Catalytic and communicating Petri nets are Turing complete. Inf. Comput. **239**, 55–70 (2014)
8. Danos, V., Krivine, J.: Reversible communicating systems. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 292–307. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28644-8_19
9. Freund, R., Kari, L., Oswald, M., Sosík, P.: Computationally universal P systems without priorities: two catalysts are sufficient. Theoret. Comput. Sci. **330**(2), 251–266 (2005)
10. Giachino, E., Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent reversibility in a tuple-based language. In: Daneshtalab, M., Aldinucci, M., Leppänen, V., Lilius, J., Brorsson, M. (eds.) PDP 2015, pp. 467–475. IEEE Computer Society (2015)
11. Ibarra, O.H.: On strong reversibility in P systems and related problems. Int. J. Found. Comput. Sci. **22**(1), 7–14 (2011)
12. Kleijn, J., Koutny, M.: A Petri net model for membrane systems with dynamic structure. Nat. Comput. **8**(4), 781–796 (2009)
13. Kleijn, J.H.C.M., Koutny, M., Rozenberg, G.: Towards a Petri net semantics for membrane systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 292–309. Springer, Heidelberg (2006). https://doi.org/10.1007/11603047_20
14. Lanese, I., Mezzina, C.A., Stefani, J.-B.: Reversing higher-order Pi. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 478–493. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_33

15. Lanese, I., Mezzina, C.A., Stefani, J.: Reversibility in the higher-order π-calculus. Theoret. Comput. Sci. **625**, 25–84 (2016)
16. Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent reversibility. Bull. EATCS (114) (2014)
17. Leporati, A., Zandron, C., Mauri, G.: Reversible P systems to simulate Fredkin circuits. Fundamenta Informaticae **74**(4), 529–548 (2006)
18. Nishida, T.Y.: Reversible P systems with symport/antiport rules. In: Paun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A. (eds.) WMC 2010, pp. 452–460 (2010)
19. Phillips, I., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. J. Logic Algebraic Methods Program. **84**(6), 781–805 (2015)
20. Pinna, G.M., Saba, A.: Modeling dependencies and simultaneity in membrane system computations. Theoret. Comput. Sci. **431**, 13–39 (2012)
21. Păun, A., Păun, G.: The power of communication: P systems with Symport/Antiport. New Gener. Comput. **20**(3), 295–306 (2002)
22. Păun, G.: Computing with membranes: an introduction. Bull. EATCS **67**, 139–152 (1999)
23. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
24. Song, T., Shi, X., Xu, J.: Reversible spiking neural P systems. Front. Comput. Sci. **7**(3), 350–358 (2013)