# Type Disciplines for Analysing Biologically Relevant Properties

Bogdan Aman

*Institute of Computer Science, Romanian Academy Blvd. Carol I no.8, 700505 Iaşi, Romania* baman@iit.tuiasi.ro

*"A.I.Cuza" University of Iaşi, Faculty of Computer Science Blvd. Carol I no.11, 700506 Iaşi, Romania*

Mariangiola Dezani-Ciancaglini Angelo Troina

*Dipartimento di Informatica, Università di Torino corso Svizzera 185, 10149 Torino, Italia* {dezani,troina}@di.unito.it

**Abstract**

The calculus of looping sequences is a formalism for describing evolution of biological systems by means of term rewriting rules. We propose to enrich this calculus with type disciplines to guarantee the soundness of reduction rules with respect to interesting biological properties.

## 1 Introduction

Biologists usually describe biological systems by mathematical means, such as differential equations. This allows them to reason on the behaviour of the described systems and to perform simulations. Mathematical modelling becomes more difficult both in specification and in analysis when the complexity of the system increases. This is one of the main motivations for the application of Computer Science formalisms to the description of biological systems [13]. Another motivation is that the use of formal means of Computer Science permits the application of analysis methods that are practically unknown to biologists, such as model checking.

Among the formalisms that either have been applied to or have been inspired by biological systems there are automata-based models [1,9], rewrite systems [7,11], and process calculi [13,14,12,6]. Automata have the advantage of allowing the direct use of many verification tools such as model checkers. Rewrite systems usually allow describing biological systems with a notation that can be easily understood by biologists. On the other hand, automata-like

*This paper is electronically published in*
*Electronic Notes in Theoretical Computer Science*
*URL:* www.elsevier.nl/locate/entcs

models and rewrite systems present, in general, problems from the point of view of compositionality. Compositionality allows studying the behaviour of a system componentwise, and is in general ensured by process calculi, included those commonly used to describe biological systems.

Milazzo et al. in [3,4,10] developed a new formalism, called Calculus of Looping Sequences (CLS for short), for describing biological systems and their evolution. CLS is based on term rewriting with some features, such as a commutative parallel composition operator, and some semantic means, such as bisimulations [4,5], which are common in process calculi. This permits to combine the simplicity of notation of rewrite systems with the advantage of a form of compositionality.

In this paper we enrich CLS with two type disciplines which allow to guarantee the soundness of reduction rules with respect to some relevant properties of biological systems. The key technical tools we use are type inference and principal typing [15], i.e. we associate to each reduction rule the minimal set of conditions an instantiation must satisfy in order to assure that applying this rule to a "correct" system we get a "correct" system as well.

To the best of our knowledge [2] is the only paper which studies a type discipline for CLS. We generalise that proposal getting in this way typability of more reduction rules.

### *1.1 Summary*

The remainder of this paper is organised as follows: Section 2 introduces the CSL, while in Section 3 we describe two type disciplines for CSL. Section 4 contains two biological examples motivating the type disciplines. Some concluding remarks end the paper.

## 2 The Calculus of Looping Sequences

In this section we recall the linked Calculus of Looping Sequences (here, we simply call it CLS) [2]. It is based on term rewriting, and hence a CLS model consists of a term and a set of rewrite rules. The term represents the structure of the modelled system, and the rewrite rules represent the events that may cause the system to evolve.

We start with defining the syntax of terms. We assume a possibly infinite alphabet $\mathcal{E}$ of symbols ranged over by $a, b, c, \ldots$ and a set of names $\mathcal{N}$ ranged over by $m, n, \ldots$.

**Definition 2.1** [Terms] *Terms* $T$ and *Sequences* $S$ of CLS are given by the grammars:

$$\begin{array}{rcl} T & ::= & S \;\big|\; (S)^L \rfloor T \;\big|\; T \,|\, T \\ S & ::= & \epsilon \;\big|\; a \;\big|\; a^n \;\big|\; S \cdot S \end{array}$$

where $a$ is any element of $\mathcal{E}$, $n$ is a name in $\mathcal{N}$ and $\epsilon$ is the empty sequence. We denote the infinite sets of terms and sequences with $\mathcal{T}$ and $\mathcal{S}$, respectively.

In CLS we have a sequencing operator $\_\cdot\_$, a looping operator $(\_)^L$, a parallel composition operator $\_\,|\,\_$, and a containment operator $\_\rfloor\_$. Sequencing can be used to concatenate elements of the alphabet $\mathcal{E}$. The empty sequence $\epsilon$ denotes the concatenation of zero symbols. A term can be either a sequence, or a looping sequence (that is the application of the looping operator to a sequence) containing another term, or the parallel composition of two terms. By the definition of terms, we have that looping and containment are always applied together, hence we can consider them as a single binary operator $(\_)^L \rfloor \_$ that applies to one sequence and one term.

The biological interpretation of the operators is the following: the main entities which occur in cells are DNA and RNA strands, proteins, membranes, and other macro-molecules. DNA strands (and similarly RNA strands) are sequences of nucleic acids, but they can be seen also at a higher level of abstraction as sequences of genes. Proteins are sequences of amino acids which usually have a very complex three-dimensional structure. In a protein there are usually (relatively) few subsequences, called domains, which actually are able to interact with other entities by means of chemical reactions. CLS sequences can model DNA/RNA strands and proteins by describing each gene or each domain with a symbol of the alphabet. The binding between two domains of two different proteins, that is the linking between two elements of two different sequences, is modelled by labelling the two symbols representing the domains with the same name. When a term represents only a part of the system we are modelling, there might be some name appearing only once. In this case, the symbols labelled with these names are linked to other symbols in other parts of the term representing the full model. Membranes are closed surfaces often interspersed with proteins, and may have a content. A closed surface can be modelled by a looping sequence. The elements (or the subsequences) of the looping sequence may represent the proteins on the membrane, and by the containment operator it is possible to specify what the membrane contains. As membranes create compartments, elements inside a looping sequences cannot be linked to elements outside. Thus, elements inside a membrane can be linked either to other elements which are also inside the membrane or to elements of the membrane itself. Other macro-molecules can be modelled as single alphabet symbols, or as sequences of their components. An element can be linked at most to another element. Note, however, that a domain able to bind with multiple partners simultaneously could be encoded by using more elements with a single binding site. Finally, juxtaposition of entities can be described by the parallel composition operator of their representations.

In CLS we may have syntactically different terms representing the same structure. Names are only labels for binding and so we consider terms modulo $\alpha$-renaming of names (denoted by $\equiv_\alpha$). We introduce also a structural equivalence relation.

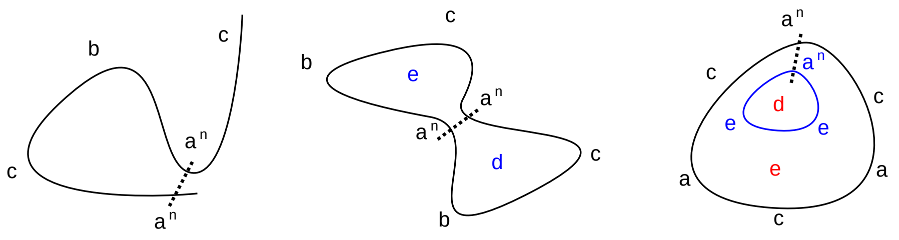

Fig. 1. Examples of CLS terms.

**Definition 2.2** [Structural equivalence] The *structural equivalence* relation $\equiv$ is the least equivalence relation on terms such that:

(i) is a congruence with respect to the operators $\_\cdot\_$, $\_\,|\,\_$, and $(\_)^L\rfloor\_$;

(ii) $\_\cdot\_$ is associative;

(iii) $\_\,|\,\_$ is commutative and associative;

(iv) $\epsilon$ is the neutral element of $\_\cdot\_$, $\_\,|\,\_$, and $(\epsilon)^L\rfloor\,\epsilon\equiv\epsilon$;

(v) looping sequences can rotate, i.e. $(S_1\cdot S_2)^L\rfloor\,T\equiv(S_2\cdot S_1)^L\rfloor\,T$.

**Example 2.3** In Figure 1 we depict some terms of the calculus illustrating the syntax of CLS. Sequences are depicted as lines, and their elements are of the same color of the line, circular lines represent looping sequences, while dotted lines depict the bindings. The first term $\mathtt{a}^{\mathtt{n}}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{a}^{\mathtt{n}}\cdot\mathtt{c}$ represents a sequence of elements (e.g., a sequence of the domains of a protein) in which two $\mathtt{a}$ elements are bound through the label $\mathtt{n}$ (e.g., two domains of the protein are bound to form a more complex structure). The second term $(\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{a}^{\mathtt{n}}\cdot\mathtt{b}\cdot\mathtt{c}\cdot\mathtt{a}^{\mathtt{n}})^L\rfloor\,(\mathtt{d}\,|\,\mathtt{e})$ represents a looping sequence (a membrane) containing elements $\mathtt{d}$ and $\mathtt{e}$ where the binding of two $\mathtt{a}$ elements may cause the membrane to divide. The last term $(\mathtt{c}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{a}^{\mathtt{n}}\cdot\mathtt{c}\cdot\mathtt{a})^L\rfloor\,((\mathtt{e}\cdot\mathtt{a}^{\mathtt{n}}\cdot\mathtt{e})^L\rfloor\,\mathtt{d}\,|\,\mathtt{e})$ represents a bubble which joins the membrane containing it with the binding on two $\mathtt{a}$ elements and prepares for endocytosis.

Rewrite rules are defined essentially as pairs of terms, in which the first term describes the portion of the system in which the event modelled by the rule may occur, and the second term describes how that portion of the system changes when the event occurs. In the terms of a rewrite rule we allow the use of variables. As a consequence, a rule will be applicable to all terms which can be obtained by properly instantiating its variables. Variables can be of three kinds: two are associated with the two different syntactic categories of terms and sequences, and one is associated with single alphabet elements. We assume a set of term variables $TV$ ranged over by $X, Y, Z, \ldots$, a set of sequence variables $SV$ ranged over by $\widetilde{x}, \widetilde{y}, \widetilde{z}, \ldots$, and a set of element variables $\mathcal{X}$ ranged over by $x, y, z, \ldots$. All these sets are pairwise disjoint and possibly infinite. We denote by $\mathcal{V}$ the set of all variables $TV\cup SV\cup\mathcal{X}$, and with $\rho$ any variable in $\mathcal{V}$. A pattern is a term which may include variables.

**Definition 2.4** [Patterns] *Patterns* $P$ and *sequence patterns* $SP$ of CLS are given by the following grammar:

$$\begin{array}{rcl} P & ::= & SP \quad \big| \quad (SP)^L \rfloor P \quad \big| \quad P \,|\, P \quad \big| \quad X \\ SP & ::= & \epsilon \quad \big| \quad a \quad \big| \quad a^n \quad \big| \quad SP{\cdot}SP \quad \big| \quad \widetilde{x} \quad \big| \quad x \quad \big| \quad x^n \end{array}$$

where $a$ is an element of $\mathcal{E}$, $n$ is a name in $\mathcal{N}$ and $X, \widetilde{x}$ and $x$ are elements of $TV, SV$ and $\mathcal{X}$, respectively. We denote with $\mathcal{P}$ the infinite set of patterns.

We assume the $\alpha$-equivalence and the structural equivalence relations to be extended to patterns. An *instantiation* is a partial function $\sigma : \mathcal{V} \rightarrow \mathcal{T}$. An instantiation must preserve the kind of variables, thus for $X \in TV$, $\widetilde{x} \in SV$ and $x \in \mathcal{X}$ we have $\sigma(X) \in \mathcal{T}, \sigma(\widetilde{x}) \in \mathcal{S}$, and $\sigma(x) \in \mathcal{E}$, respectively. Given $P \in \mathcal{P}$, with $P\sigma$ we denote the term obtained by replacing each occurrence of each variable $\rho \in \mathcal{V}$ appearing in $P$ with the corresponding term $\sigma(\rho)$. With $\Sigma$ we denote the set of all the possible instantiations, and, given $P \in \mathcal{P}$, with $Var(P)$ we denote the set of variables appearing in $P$.

We can now define rewrite rules.

**Definition 2.5** [Rewrite Rules] A *rewrite rule* is a pair of patterns $(P_1, P_2)$, denoted with $P_1 \mapsto P_2$, where $P_1, P_2 \in \mathcal{P}$, $P_1 \not\equiv \epsilon$ and such that $Var(P_2) \subseteq Var(P_1)$.

A rewrite rule $P_1 \mapsto P_2$ states that a term $P_1\sigma$, obtained by instantiating variables in $P_1$ by some instantiation function $\sigma$, can be transformed into the term $P_2\sigma$. We define the semantics of CLS as a transition system, in which states correspond to terms, and transitions correspond to rule applications.

The semantics of CLS is defined by resorting to the notion of contexts.

**Definition 2.6** [Contexts] *Contexts* $C$ are defined as:

$$C ::= \square \quad \big| \quad C \,|\, T \quad \big| \quad T \,|\, C \quad \big| \quad (S)^L \rfloor C$$

where $T \in \mathcal{T}$ and $S \in \mathcal{S}$. The context $\square$ is called the *empty context*. We denote with $\mathcal{C}$ the infinite set of contexts.

By definition, every context contains a single hole $\square$. Let us assume $C, C' \in \mathcal{C}$. With $C[T]$ we denote the term obtained by replacing $\square$ with $T$ in $C$; with $C[C']$ we denote context composition, whose result is the context obtained by replacing $\square$ with $C'$ in $C$. The $\alpha$-equivalence and the structural equivalence can be easily extended to contexts, namely $C \equiv_\alpha C'$ if $C[\epsilon] \equiv_\alpha C'[\epsilon]$, and similarly for $\equiv$.

Rewrite rules can be applied to terms only if they occur in a legal context. Note that the general form of rewrite rules does not permit to have sequences as contexts. A rewrite rule introducing a parallel composition on the right hand side (as $a \mapsto b \,|\, c$) applied to an element of a sequence (e.g., $m \cdot a \cdot m$) would result into a syntactically incorrect term (in this case $m \cdot (b \,|\, c) \cdot m$). To modify a sequence, a pattern representing the whole sequence must appear in the rule. For example, rule $a{\cdot}\widetilde{x} \mapsto a \,|\, \widetilde{x}$ can be applied to any sequence starting with element $a$, and, hence, the term $a \cdot b$ can be rewritten as $a \,|\, b$, and the term $a{\cdot}b{\cdot}c$ can be rewritten as $a \,|\, b{\cdot}c$.

The semantics of CLS is defined as follows. We denote by $\mathcal{O}(T)$, $\mathcal{T}(T)$ the set of names which occur once or twice in $T$, respectively. We define $\mathcal{O}(C) = \mathcal{O}(C[\epsilon])$ and $\mathcal{T}(C) = \mathcal{T}(C[\epsilon])$.

**Definition 2.7** [Semantics] Given a finite set of rewrite rules $\mathcal{R}$, the *semantics* of CLS is the least relation closed with respect to $\equiv$ and satisfying the following inference rule:

$$\frac{\begin{array}{c} P_1 \mapsto P_2 \in \mathcal{R} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma \quad C \in \mathcal{C} \\ C' \equiv_\alpha C \quad T \equiv_\alpha P_2\sigma \quad \mathcal{O}(C') \cap \mathcal{T}(T) = \mathcal{T}(C') \cap \mathcal{O}(T) = \mathcal{T}(C') \cap \mathcal{T}(T) = \emptyset \end{array}}{C[P_1\sigma] \to C'[T]}$$

$\alpha$-renaming is used to avoid name clashes, since it is easy to verify that for any $C, P_2, \sigma$ there are $C', T$ such that $C' \equiv_\alpha C$, $T \equiv_\alpha P_2\sigma$, and $\mathcal{O}(C') \cap \mathcal{T}(T) = \mathcal{T}(C') \cap \mathcal{O}(T) = \mathcal{T}(C') \cap \mathcal{T}(T) = \emptyset$. As usual we denote with $\to^*$ the reflexive and transitive closure of $\to$.

Given a set of rewrite rules $\mathcal{R}$, the behaviour of a term $T$ is the tree of terms to which $T$ may reduce.

## 3 Type Disciplines

### 3.1 A Type Discipline for Safe Bindings

The use of links may cause some problems. Consider for example the two terms $a^n \,|\, b^n \,|\, c^n$ and $a^n \,|\, (b{\cdot}c)^L \rfloor d^n$. In the first one, the same name $n$ is used to label more than two elements; in the second term a name is used to link an element outside a membrane and an element inside it. To avoid this kind of situations, a notion of well-formed terms is introduced in [2] stating that a term is well formed if and only if a labelling name occurs no more than twice within the term, and that two occurrences of a label are always within the same compartment. A type discipline for checking the well formedness of terms and patterns is proposed in [2]. Here we propose a generalisation of that type discipline in two respects:

- we classify elements with basic types and we assure that only elements of the same basic types are linked;
- we relax the restrictions on the shapes of the rewrite rules and of the instantiations.

To this aim we require atomic elements in $\mathcal{E}$ to be of some basic type $\mathtt{t}$. Intuitively, given a molecule represented by an element in $\mathcal{E}$, we associate to it a type $\mathtt{t}$ which specifies the kind of the molecule and the kind of bindings the molecule can create. We assume a fixed typing $\Gamma_0$ for the elements in $\mathcal{E}$.

$$\Gamma \vdash \epsilon : (\emptyset, \emptyset)\ (\epsilon) \qquad \frac{a : \mathtt{t} \in \Gamma_0}{\Gamma \vdash a : (\emptyset, \emptyset)}\ (a) \qquad \frac{a : \mathtt{t} \in \Gamma_0}{\Gamma \vdash a^n : (\emptyset, \{a : \mathtt{t}\})}\ (a^n)$$

$$\Gamma, x : \mathtt{t} \vdash x : (\emptyset, \emptyset)\ (x) \qquad \Gamma, x : \mathtt{t} \vdash x^n : (\emptyset, \{a : \mathtt{t}\})\ (x^n) \qquad \Gamma, \eta : (\mathtt{N}, \mathtt{L}) \vdash \eta : (\mathtt{N}, \mathtt{L})\ (\eta)$$

$$\frac{\Gamma \vdash SP : (\mathtt{N}, \mathtt{L}) \quad \Gamma \vdash SP' : (\mathtt{N}', \mathtt{L}') \quad \mathtt{N} \cap \mathtt{N}' = \mathtt{N} \cap dom(\mathtt{L}') = \mathtt{N}' \cap dom(\mathtt{L}) = \emptyset \quad \mathtt{L} \bowtie \mathtt{L}'}{\Gamma \vdash SP \cdot SP' : (\mathtt{N} \cup \mathtt{N}' \cup (dom(\mathtt{L}) \cap dom(\mathtt{L}')), \mathtt{L} \uplus \mathtt{L}')}\ (seq)$$

$$\frac{\Gamma \vdash P : (\mathtt{N}, \mathtt{L}) \quad \Gamma \vdash P' : (\mathtt{N}', \mathtt{L}') \quad \mathtt{N} \cap \mathtt{N}' = \mathtt{N} \cap dom(\mathtt{L}') = \mathtt{N}' \cap dom(\mathtt{L}) = \emptyset \quad \mathtt{L} \bowtie \mathtt{L}'}{\Gamma \vdash P \mid P' : (\mathtt{N} \cup \mathtt{N}' \cup (dom(\mathtt{L}) \cap dom(\mathtt{L}')), \mathtt{L} \uplus \mathtt{L}')}\ (parcomp)$$

$$\frac{\Gamma \vdash SP : (\mathtt{N}, \mathtt{L}) \quad \Gamma \vdash P : (\mathtt{N}', \mathtt{L}') \quad \mathtt{N} \cap \mathtt{N}' = \mathtt{N} \cap dom(\mathtt{L}') = \mathtt{N}' \cap dom(\mathtt{L}) = \emptyset \quad \mathtt{L}' \subseteq \mathtt{L}}{\Gamma \vdash (SP)^L \rfloor P : (\mathtt{N} \cup dom(\mathtt{L}'), \mathtt{L} \setminus \mathtt{L}')}\ (loop)$$

Fig. 2. Typing rules for safe bindings

We use $\eta \in SV \cup TV$ to denote either sequence or term variables. With N we denote a finite set of untyped names:

$$\mathtt{N} ::= \emptyset \mid \mathtt{N}, n$$

while with L we denote a finite set of typed names:

$$\mathtt{L} ::= \emptyset \mid \mathtt{L}, n : \mathtt{t}$$

Intuitively, give a pattern $P$, we can associate to $P$ a set of names N which contains all the names used to create closed point to point bindings. A pattern may however also contain some names which do not bind another molecule in $P$ but may bind somewhere else in the environment. Thus, we may associate to a pattern $P$ a set L of typed names. We do not need to keep track of types for closed link, but, for open links, we should guarantee that a molecule of some type is bound, somewhere else in the environment, with a molecule of the same type. To sum up we associate to a pattern a *pair type* of the shape $(\mathtt{N}, \mathtt{L})$. We associate pair types also to sequences.

We define the domain of a set of typed names L as

$$dom(\mathtt{L}) = \{n \mid n : \mathtt{t} \in \mathtt{L}\}.$$

We say that two typed set of names L and L′ are *compatible* (written $\mathtt{L} \bowtie \mathtt{L}'$) if and only if whenever $n : \mathtt{t} \in \mathtt{L}$ and $n : \mathtt{t}' \in \mathtt{L}'$, then it holds $\mathtt{t} = \mathtt{t}'$. The disjoint union of L and L′ is defined as

$$\mathtt{L} \uplus \mathtt{L}' = \{n : \mathtt{t} \in \mathtt{L} \wedge n \notin dom(\mathtt{L}')\} \cup \{n : \mathtt{t}' \in \mathtt{L}' \wedge n \notin dom(\mathtt{L})\}.$$

With the following grammar we define *basis* $\Gamma$, which maps element variables to basic types, and maps sequence and term variables to pair types:

$$\Gamma ::= \emptyset \mid \Gamma, \nu : \mathtt{t} \mid \Gamma, \eta : (\mathtt{N}, \mathtt{L})$$

The type discipline to check safe bindings, namely, to avoid not well-formed bindings, is defined by the typing rules in Figure 2.

Rules $(\epsilon)$-$(a)$-$(x)$: any basis types with N and L empty sets, the term $\epsilon$ and any elementary object without binding labels. Rules $(a^n)$-$(x^n)$: an elementary object with a binding label $n$ gets typed with $\mathtt{N}$ empty (there are no labels defining a closed link) and with $\mathtt{L} = \{n : \mathtt{t}\}$ (there is an open link represented by label $n$ of type $\mathtt{t}$). Rule ($\eta$): complex sequences and terms may contain both closed links and open links.

Rule (*seq*): when putting two sequences together, the names representing the closed links should not appear in any other binding. Open links in the two sequences to be join can form a closed link if they have the same label (since we require compatibility between $\mathtt{L}$ and $\mathtt{L}'$, the labels closing each open link are of the same type). In the resulting sequences, the labels which got closed are removed form $\mathtt{L}$ and $\mathtt{L}'$ and added to the final set of labels representing closed links. Rule (*parcomp*): similarly as what happens for Rule (*seq*), putting two patterns in parallel may allow to close some of the links which are open in the the two patterns in isolation.

Rule (*loop*): we can put a pattern $P$ inside a looping sequence $SP$ only when all the open links of $P$ are closed. This is because if $P$ gets inside a compartment (represented by the looping sequence) it cannot interact any more with the environment. Thus, if $P$ has some open link, it should be bound with equal open links present on the looping sequence $SP$, which now represents the only environment surrounding $P$. For this to be done, we require that the set of open links of $P$ is a subset of the set of open links of $SP$ (all the open links in $P$ can be closed by $SP$).

Rewrite rules may modify the status of the bindings by creating new closed links or destroying some of them. We require, however, that rewrite rules do not change the status of open bindings, which are assumed to be closed by the environment and represent only a partial state of the system.

**Definition 3.1** [$\Gamma$-Safe Rules] A rewrite rule $P \mapsto P'$ is $\Gamma$-*safe* (notation $P \mapsto P' \in \mathcal{R}_\Gamma$) if $\Gamma \vdash P : (\mathtt{N}, \mathtt{L})$ and $\Gamma \vdash P' : (\mathtt{N}', \mathtt{L})$ for some $\mathtt{N}, \mathtt{N}', \mathtt{L}$.

An instantiation $\sigma$ *agrees* with a basis $\Gamma$ (notation $\sigma \in \Sigma_\Gamma$) if $x : \mathtt{t} \in \Gamma$ implies $\sigma(x) : \mathtt{t} \in \Gamma$ and $\eta : (\mathtt{N}', \mathtt{L}) \in \Gamma$ implies $\Gamma \vdash \sigma(\eta) : (\mathtt{N}, \mathtt{L})$. We can safely apply a $\Gamma$-safe rule to a term only if the involved instatiation agrees with $\Gamma$. In this case we denote by $\xrightarrow{\Gamma}$ the so obtained reduction. More formally:

**Definition 3.2** [$\Gamma$-Typed Semantics] Given a finite set of rewrite rules $\mathcal{R}$, the $\Gamma$-*typed semantics* of CLS is the least relation closed with respect to $\equiv$ and satisfying the following inference rule:

$$\frac{\begin{array}{c} P_1 \mapsto P_2 \in \mathcal{R}_\Gamma \qquad P_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma_\Gamma \qquad C \in \mathcal{C} \\ C' \equiv_\alpha C \quad T \equiv_\alpha P_2\sigma \quad \mathcal{O}(C') \cap \mathcal{T}(T) = \mathcal{T}(C') \cap \mathcal{O}(T) = \mathcal{T}(C') \cap \mathcal{T}(T) = \emptyset \end{array}}{C[P_1\sigma] \xrightarrow{\Gamma} C'[T]}$$

As expected $\xrightarrow{\Gamma}$ reduction preserves typing under the basis $\Gamma$.

**Theorem 3.3** *If* $\Gamma \vdash T : (\mathtt{N}, \mathtt{L})$ *and* $T \xrightarrow{\Gamma} T'$, *then* $\Gamma \vdash T' : (\mathtt{N}', \mathtt{L})$ *for some* $\mathtt{N}'$.

$$\vdash \epsilon : \emptyset; (\emptyset, \emptyset); \emptyset \qquad \frac{a : \mathtt{t} \in \Gamma_0}{\vdash a : \emptyset; (\emptyset, \emptyset); \emptyset} \qquad \frac{a : \mathtt{t} \in \Gamma_0}{\vdash a^n : \emptyset; (\emptyset, \{n : \mathtt{t}\}); \emptyset}$$

$$\vdash x : \{x : \varphi_x\}; (\emptyset, \emptyset); \emptyset \qquad \vdash x^n : \{x : \varphi_x\}; (\emptyset, \{n : \varphi_x\}); \emptyset$$

$$\vdash \eta : \{\eta : (\phi_\eta, \psi_\eta)\}; (\phi_\eta, \psi_\eta); \emptyset$$

$$\frac{\vdash SP : \Theta; (\Phi, \Psi); \Xi \qquad \vdash SP' : \Theta'; (\Phi', \Psi'); \Xi'}{\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Phi \cup \Phi' \cup (dom(\Psi) \cap dom(\Psi')), \Psi \uplus \Psi'); \Xi''}$$
where $\Xi'' = \Xi \cup \Xi' \cup (\Phi \cap \Phi' = \Phi \cap dom(\Psi') = \Phi' \cap dom(\Psi) = \emptyset) \cup (\Psi \bowtie \Psi')$

$$\frac{\vdash P : \Theta; (\Phi, \Psi); \Xi \qquad \vdash P' : \Theta'; (\Phi', \Psi'); \Xi'}{\vdash P \mid P' : \Theta \cup \Theta'; (\Phi \cup \Phi' \cup (dom(\Psi) \cap dom(\Psi')), \Psi \uplus \Psi'); \Xi''}$$
where $\Xi'' = \Xi \cup \Xi' \cup (\Phi \cap \Phi' = \Phi \cap dom(\Psi') = \Phi' \cap dom(\Psi) = \emptyset) \cup (\Psi \bowtie \Psi')$

$$\frac{\vdash SP : \Theta; (\Phi, \Psi); \Xi \qquad \vdash P : \Theta'; (\Phi', \Psi'); \Xi'}{\vdash (SP)^L \rfloor P : \Theta \cup \Theta'; (\Phi \cup dom(\Psi'), \Psi \setminus \Psi'); \Xi''}$$
where $\Xi'' = \Xi \cup \Xi' \cup (\Phi \cap \Phi' = \Phi \cap dom(\Psi') = \Phi' \cap dom(\Psi) = \emptyset) \cup (\Psi' \subseteq \Psi)$

Fig. 3. Inference Rules for Principal Typing

Note that we generalised the typability of [2]. For example let $\Gamma = \{a : \mathtt{t}, b : \mathtt{t}, \widetilde{x} : (\emptyset, \{1 : \mathtt{t}\})\}$, $\sigma(\widetilde{x}) = b^1$, $a^1 \mid \widetilde{x} \mapsto a^1 \mid a^1$: we get $a^1 \mid b^1 \xrightarrow{\Gamma} a^1 \mid a^1$ and this example cannot be dealt with the type discipline of [2].

In order to infer which rewriting rules are $\Gamma$-safe the machinery of *principal typing* [15] is handy. We convene that for each variable $x \in \mathcal{X}$ there is an *e-type variable* $\varphi_x$ ranging over basic types, and for each variable $\eta \in SV \cup TV$ there are two variables $\phi_\eta$, $\psi_\eta$ (called *u-type variable* and *t-type variable*) ranging over sets of untyped and typed names, respectively. Moreover we convene that $\Phi$ ranges over unions of sets of untyped names and u-type variables, and $\Psi$ ranges over unions of sets of typed names and t-type variables.
A *basis scheme* $\Theta$ is a map from atomic variables to their e-types, and from sequence and term variables to pairs of their u-type variables and t-type variables:

$$\Theta \ ::= \ \emptyset \ \Big| \ \Theta, x : \varphi_x \ \Big| \ \Theta, \eta : (\phi_\eta, \psi_\eta)$$

The rules for inferring principal typing use judgements of the shape:

$$\vdash P : \Theta; (\Phi, \Psi); \Xi \qquad \vdash SP : \Theta; (\Phi, \Psi); \Xi$$

where $\Theta$ is the *principal basis* in which $P$ ($SP$) is well formed, $(\Phi, \Psi)$ is the *principal type* of $P$ ($SP$), and $\Xi$ is the set of conditions which should be satisfied when building up $P$ ($SP$). Figure 3 gives these inference rules.

Soundness and completeness of our inference rules can be stated as usual. A *type mapping* maps e-type variables to basic types, u-type variables to sets of names and t-type variables to sets of typed names. A type mapping $\mathsf{m}$ *satisfies* a set of constraints $\Xi$ if all constraints in $\mathsf{m}(\Xi)$ hold true.

**Theorem 3.4 (Soundness of Type Inference)** (i) *If* $\vdash SP : \Theta; (\Phi, \Psi); \Xi$

*and* $\mathsf{m}$ *is a type mapping which satisfies* $\Xi$*, then* $\mathsf{m}(\Theta) \vdash SP : (\mathsf{m}(\Phi), \mathsf{m}(\Psi))$.

(ii) *If* $\vdash P : \Theta; (\Phi, \Psi); \Xi$ *and* $\mathsf{m}$ *is a type mapping which satisfies* $\Xi$*, then* $\mathsf{m}(\Theta) \vdash P : (\mathsf{m}(\Phi), \mathsf{m}(\Psi))$.

**Theorem 3.5 (Completeness of Type Inference)** (i) *If* $\vdash SP : \Theta; (\Phi, \Psi); \Xi$ *and* $\Gamma \vdash SP : (\mathtt{N}, \mathtt{L})$*, then there is a type mapping* $\mathsf{m}$ *that satisfies* $\Xi$ *and such that* $\Gamma \supseteq \mathsf{m}(\Theta)$, $\mathtt{N} = \mathsf{m}(\Phi)$, $\mathtt{L} = \mathsf{m}(\Psi)$.

(ii) *If* $\vdash P : \Theta; (\Phi, \Psi); \Xi$ *and* $\Gamma \vdash P : (\mathtt{N}, \mathtt{L})$*, then there is a type mapping* $\mathsf{m}$ *that satisfies* $\Xi$ *and such that* $\Gamma \supseteq \mathsf{m}(\Theta)$, $\mathtt{N} = \mathsf{m}(\Phi)$, $\mathtt{L} = \mathsf{m}(\Psi)$.

We conclude this subsection by putting our inference rules at work in order to assure safety of reduction rules.

Each rewrite rule induces a set of constraints $\Xi$ which takes into account the principal typing of the l.h.s. and of the r.h.s. of the rule and the notion of safety of a rule with respect to a given basis (Definition 3.1). This is formalised in the following definition.

**Definition 3.6** [$\Xi$-SuperSafe Rules] A rewrite rule $P \mapsto P'$ is $\Xi$-*SuperSafe* (notation $P \mapsto P' \in \mathcal{R}_\Xi$) if $\vdash P : \Theta'; (\Phi, \Psi); \Xi'$ , $\vdash P' : \Theta''; (\Phi', \Psi'); \Xi''$, and $\Xi = \{\Psi = \Psi'\} \cup \Xi' \cup \Xi'' \cup \{\tau = \tau' | \lambda : \tau \in \Theta \& \lambda : \tau' \in \Theta'\}$ for some $\Theta, \Phi, \Psi, \Xi', \Theta', \Phi', \Psi', \Xi''$.

We can show that SuperSafety exactly captures the notion of safety via type mappings which agree with the current set of constraints.

**Theorem 3.7 (Soundness and Completeness of $\Xi$-SuperSafety)** *If* $\mathsf{m}$ *is a type mapping we define* $\Gamma_{\mathsf{m}} = \{x : \mathsf{m}(\varphi_x)\} \cup \{\eta : (\mathsf{m}(\phi_\eta), \mathsf{m}(\psi_\eta))\}$.

(i) *If* $P \mapsto P' \in \mathcal{R}_\Xi$ *and* $\mathsf{m}$ *is a type mapping that satisfies* $\Xi$*, then* $P \mapsto P' \in \mathcal{R}_{\Gamma_{\mathsf{m}}}$.

(ii) *If* $P \mapsto P' \in \mathcal{R}_\Xi$ *and* $P \mapsto P' \in \mathcal{R}_\Gamma$*, then there is a type mapping* $\mathsf{m}$ *that satisfies* $\Xi$ *and such that* $\Gamma \supseteq \Gamma_{\mathsf{m}}$.

Similarly to what we did for safety of rules with SuperSafety we need to generalise the agreement of instantion. An instantiation $\sigma$ *superAgrees* with $\Xi$ (notation $\sigma \in \Sigma_\Xi$) if there is a type mapping $\mathsf{m}$ that satisfies $\Xi$ and such that $\sigma \in \Sigma_{\Gamma_{\mathsf{m}}}$. Given $\Xi, \sigma$ we can build an $\mathsf{m}$ which makes $\sigma$ superAgree with $\Xi$ (whenever it exists) as follows:

(i) if $\sigma(x) : \mathtt{t} \in \Gamma_0$ then $\mathsf{m}(\varphi_x) = \mathtt{N}$;

(ii) if $\vdash \sigma(\eta) : (\mathtt{N}, \mathtt{L})$ then $\mathsf{m}(\phi_\eta) = \mathtt{N}$ and $\mathsf{m}(\psi_\eta) = \mathtt{L}$.

It is easy to verify that $\sigma \in \Sigma_\Xi$ if $\mathsf{m}(\Xi)$ holds for such an $\mathsf{m}$.

We can give then our last formulation of the reduction rules, which allows us to check their safety, as stated in the following theorem.

**Definition 3.8** [$\Xi$-Typed Semantics] Given a finite set of rewrite rules $\mathcal{R}$, the $\Xi$-*typed semantics* of CLS is the least relation closed with respect to $\equiv$ and satisfying the following inference rule:

$$\frac{P_1 \mapsto P_2 \in \mathcal{R}_\Xi \qquad P_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma_\Xi \qquad C \in \mathcal{C} \qquad C' \equiv_\alpha C \quad T \equiv_\alpha P_2\sigma \quad \mathcal{O}(C') \cap \mathcal{T}(T) = \mathcal{T}(C') \cap \mathcal{O}(T) = \mathcal{T}(C') \cap \mathcal{T}(T) = \emptyset}{C[P_1\sigma] \xrightarrow{\Xi} C'[T]}$$

**Theorem 3.9 (Soundness and Completeness of the Ξ-Typed Semantics)**

(i) *If* $T \xrightarrow{\Xi} T'$ *and* $\mathsf{m}$ *is a type mapping that satisfies* $\Xi$*, then* $T \xrightarrow{\Gamma_\mathsf{m}} T'$.

(ii) *If* $T \xrightarrow{\Xi} T'$ *and* $T \xrightarrow{\Gamma} T'$*, then there is a type mapping* $\mathsf{m}$ *that satisfies* $\Xi$ *and such that* $\Gamma \supseteq \Gamma_\mathsf{m}$.

We can also use type disciplines in order to prescribe that reduction rules can be applied only if some typing conditions are satisfied. This is exemplified in Subsection 4.3 for the type discipline of Subsection 3.2.

## *3.2 A Type Discipline for Present/Required/Excluded Elements*

In this subsection we consider terms, sequences and patterns without names, since we are interested in properties orthogonal to names. We use $\mathtt{t}$ to denote a basic type, and $\mathtt{P}$,$\mathtt{R}$,$\mathtt{E}$ to denote sets of basic types. We consider only *local* properties: elements influence each other if they are either in the same compartment or they contain each other.

Types are triples of sets of basic types $(\mathtt{P}, \mathtt{R}, \mathtt{E})$: the set $\mathtt{P}$ of *present* elements, the set $\mathtt{R}$ of *required* elements, and the set $\mathtt{E}$ of *excluded* elements. A type $(\mathtt{P}, \mathtt{R}, \mathtt{E})$ is *well formed* if $\mathtt{P} \cap \mathtt{R} = \mathtt{P} \cap \mathtt{E} = \mathtt{R} \cap \mathtt{E} = \emptyset$.
Basis are defined by:

$$\Delta \;::=\; \emptyset \;\;\big|\;\; \Delta, x : (\{\mathtt{t}\}, \mathtt{R}, \mathtt{E}) \;\;\big|\;\; \Delta, \eta : (\mathtt{P}, \mathtt{R}, \mathtt{E})$$

$$\Delta \vdash \epsilon : (\emptyset, \emptyset, \emptyset) \qquad \frac{a : (\{\mathtt{t}\}, \mathtt{R}, \mathtt{E}) \in \Gamma_0}{\Delta \vdash a : (\{\mathtt{t}\}, \mathtt{R}, \mathtt{E})}$$

$$\Delta, x : (\{\mathtt{t}\}, \mathtt{R}, \mathtt{E}) \vdash x : (\{\mathtt{t}\}, \mathtt{R}, \mathtt{E}) \qquad \Delta, \eta : (\mathtt{P}, \mathtt{R}, \mathtt{E}) \vdash \eta : (\mathtt{P}, \mathtt{R}, \mathtt{E})$$

$$\frac{\Delta \vdash SP : (\mathtt{P}, \mathtt{R}, \mathtt{E}) \quad \Delta \vdash SP' : (\mathtt{P}', \mathtt{R}', \mathtt{E}') \quad \mathtt{P} \cap \mathtt{E}' = \mathtt{P}' \cap \mathtt{E} = \mathtt{R} \cap \mathtt{E}' = \mathtt{R}' \cap \mathtt{E} = \emptyset}{\Delta \vdash SP{\cdot}SP' : (\mathtt{P} \cup \mathtt{P}', (\mathtt{R} \cup \mathtt{R}') \setminus (\mathtt{P} \cup \mathtt{P}'), \mathtt{E} \cup \mathtt{E}')}$$

$$\frac{\Delta \vdash P : (\mathtt{P}, \mathtt{R}, \mathtt{E}) \quad \Delta \vdash P' : (\mathtt{P}', \mathtt{R}', \mathtt{E}') \quad \mathtt{P} \cap \mathtt{E}' = \mathtt{P}' \cap \mathtt{E} = \mathtt{R} \cap \mathtt{E}' = \mathtt{R}' \cap \mathtt{E} = \emptyset}{\Delta \vdash P \,|\, P' : (\mathtt{P} \cup \mathtt{P}', (\mathtt{R} \cup \mathtt{R}') \setminus (\mathtt{P} \cup \mathtt{P}'), \mathtt{E} \cup \mathtt{E}')}$$

$$\frac{\Delta \vdash SP : (\mathtt{P}, \mathtt{R}, \mathtt{E}) \quad \Delta \vdash P : (\mathtt{P}', \mathtt{R}', \mathtt{E}') \quad \mathtt{P} \cap \mathtt{E}' = \mathtt{P}' \cap \mathtt{E} = \mathtt{R} \cap \mathtt{E}' = \mathtt{R}' \setminus \mathtt{P} = \emptyset}{\Delta \vdash (SP)^L \rfloor P : (\mathtt{P}, \mathtt{R} \setminus \mathtt{P}', \mathtt{E})}$$

Fig. 4. Typing rules for Present/Required/Excluded Elements

With abuse of notation, we say $(\mathtt{P},\mathtt{R},\mathtt{E}) \in \Delta$ when $x : (\{\mathtt{t}\},\mathtt{R},\mathtt{E}) \in \Delta$ or $\eta : (\mathtt{P},\mathtt{R},\mathtt{E}) \in \Delta$. A basis $\Delta$ is *well formed* if:

- $(\mathtt{P},\mathtt{R},\mathtt{E}) \in \Delta \implies (\mathtt{P},\mathtt{R},\mathtt{E})$ is well formed;
- $(\mathtt{P},\mathtt{R},\mathtt{E}) \in \Delta \;\&\; (\mathtt{P}',\mathtt{R}',\mathtt{E}') \in \Delta \;\&\; \mathtt{P} \subseteq \mathtt{P}' \implies \mathtt{R} \setminus \mathtt{P}' \subseteq \mathtt{R}' \;\&\; \mathtt{E} \subseteq \mathtt{E}'$.

We check the safety of terms and sequences using the typing rules of Figure 4. It is easy to verify that if we start from well-formed environments, then we produce only well-formed environments and well-formed types.

In order to assure safety of reduction rules with respect to this type discipline one can design a type inference system and introduce a notion of Super-Safe rules for a set of constraints as we did for the previous type discipline in Subsection 3.1.

## 4 Case Studies

### 4.1 Simple Example

Consider the evolution rule:

$$a^1 \mid x \mapsto a \mid x^1.$$

If $a : \mathtt{t} \in \Gamma_0$, then the l.h.s. of this rule has the following principal typing:

$$\frac{\vdash a^1 : \emptyset; (\emptyset, \{1 : \mathtt{t}\}); \emptyset \qquad \vdash x : \{x : \phi_x\}; (\emptyset, \emptyset); \emptyset}{\vdash a^1 \mid x : \{x : \phi_x\}; (\emptyset, \{1 : \mathtt{t}\}); \emptyset}$$

while the r.h.s. has the following principal typing:

$$\frac{\vdash a : \emptyset; (\emptyset, \emptyset); \emptyset \qquad \vdash x^1 : \{x : \phi_x\}; (\emptyset, \{1 : \phi_x\}); \emptyset}{\vdash a \mid x^1 : \{x : \phi_x\}; (\emptyset, \{1 : \phi_x\}); \emptyset}$$

The rule is then $\Xi$-SuperSafe with $\Xi = \{\{1 : \mathtt{t}\} = \{1 : \phi_x\}\}$. A type mapping satisfying $\Xi$ is then clearly $\mathsf{m}(\phi_x) = \mathtt{t}$. We conclude that this rule can be safely applied for all instantiations which map $x$ to a basic element of type $\mathtt{t}$.

### 4.2 Fusion

Membrane fusion is the process by which a vesicle membrane incorporates its components into the target membrane and releases its cargo into the lumen of the organelle or, in the case of secretion, into the extracellular medium.

Different steps in membrane fusion are distinguished. First, the vesicle and the target membrane mutually identify each other. Then, proteins from both membranes interact with one another to form stable complexes and bring the two membranes into close apposition, resulting in the docking of the vesicle to the target membrane. Finally, considerable energy needs to be supplied to force the membranes to fuse, since the low-energy organizationin which the hydrophobic tails of the phospholipids are kept away from water while the

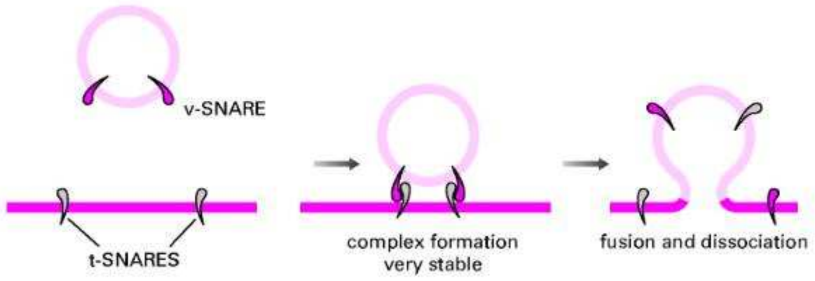

Fig. 5. SNAREs and vesicle fusion.

hydrophilic head groups are in an aqueous mediummust be disrupted, even if only briefly, as the vesicle and target membranes distort and then fuse. Each type of vesicle must only dock with and fuse with the correct target membrane, otherwise the protein constituents of all the different organelles would become mixed with each other and with the plasma membrane.

Our understanding of the molecular processes leading to membrane fusion is only just beginning to take shape, but our current understanding is that two types of proteins, called SNARES and Rab family GTPases work together to achieve this. SNARES located on the vesicles (v-SNARES) and on the target membranes (t-SNARES) interact to form a stable complex that holds the vesicle very close to the target membrane (Fig. 5). Not all vSNARES can interact with all tSNARES, so SNARES provide a first level of specificity. So far, over 50 members of the Rab family have been identified in mammalian cells, and each seems to be found at one particular site where it regulates one specific transport event, thus controlling which vesicle fuses with which target.

Consider the initial configuration:

$$Initial = (a_v^1 \widetilde{x} b_v^2)^L \rfloor \widetilde{z} \mid (a_t^1 b_t^2 \widetilde{y})^L \rfloor \widetilde{w}$$

where the first component represents the vesicle, while the second component represents the target membrane, and the indexes only distinguish different occurrences of the same protein.

This evolves to the configuration:

$$Final = (a_t a_v \widetilde{x} b_t b_v \widetilde{y})^L \rfloor (\widetilde{w} \mid \widetilde{z})$$

by applying the evolution rule:

$$Initial \mapsto Final$$

If $a : \mathtt{t}_a \in \Gamma_0$, then the first step of the type inference for $Initial$ is:

$$\frac{\vdash a_v^1 : \emptyset; (\emptyset, \{1 : \mathtt{t}_a\}); \emptyset \qquad \vdash \widetilde{x} : \{\widetilde{x} : (\phi_{\widetilde{x}}, \psi_{\widetilde{x}})\}; (\phi_{\widetilde{x}}, \psi_{\widetilde{x}}); \emptyset}{\vdash a_v^1 \widetilde{x} : \{\widetilde{x} : (\phi_{\widetilde{x}}, \psi_{\widetilde{x}})\}; (\phi_{\widetilde{x}} \cup (dom(\psi_{\widetilde{x}}) \cap \{1\}), \{1 : t_1\} \uplus \psi_{\widetilde{x}}); \Xi}$$

where $\Xi = (\phi_{\widetilde{x}} \cap \{1\} = \emptyset) \cup (\{1 : t_1\} \bowtie \psi_{\widetilde{x}})$. The set $\Xi$ prescripts that $\widetilde{x}$ should not contain two links labelled 1 and if it contains a link labelled 1 then this link has type $\mathtt{t}_a$.

The whole set of constraints which assures the safety of the application for this rule requires that:

(i) $\widetilde{x}$, $\widetilde{z}$, $\widetilde{y}$, and $\widetilde{w}$ do not have links labelled 1 or 2;

(ii) the open links offered by $\widetilde{z}$ are contained in the open links offered by $\widetilde{x}$;

(iii) the open links offered by $\widetilde{w}$ are contained in the open links offered by $\widetilde{y}$.

These are the conditions we need to check when the variables $\widetilde{x}$, $\widetilde{z}$, $\widetilde{y}$, and $\widetilde{w}$ are instantiated in order to safely apply this reduction rule. Similar checking steps can be performed if we consider that the membranes that fuse contain term variables $Z$ and $W$ instead of sequence variables $\widetilde{z}$ and $\widetilde{w}$.

### *4.3 Fusion with Promoters and Inhibitors*

Membrane fusion is a key event in a variety of important biological processes, including exocytosis, endocytosis, synaptic transmission, fertilization, and viral infection. Several investigators [8] have found that the exogenous addition of specific lipids can modulate the fusion between biological membranes or lipid vesicles. In many fusion events, fusion is reversibly inhibited by the exogenous addition of lysophosphatidylcholine (lysoPC) between apposing membranes. On the other hand, the exogenous addition of glycerol monoleate (GMO), oleic acid (OA), or arachidonic acid (AA) has been shown to promote cell-cell fusion.

Considering that we have a cell, which does not contain objects inside, that is ready to fusion with another cell. Then to the reduction rule:

$$(\tilde{x})^L \rfloor \epsilon \mid (\tilde{y})^L \rfloor \tilde{z} \mapsto (\tilde{x}\tilde{y})^L \rfloor \tilde{z}$$

we can add the condition that the in type of the l.h.s.:

(i) the type of promoters is present or required;

(ii) the type of inhibitors is excluded.

## 5 Conclusions

This paper is a first step toward the application of principal typing to the safety of system transformations which model biological phenomena. We plan to investigate type disciplines assuring different properties for CLS and to apply this approach to other calculi for describing evolution of biological systems, in particular to P systems.

## References

[1] Alur, R., Belta, C., Ivancic, F., Kumar, V., Mintz, M., Pappas, G.J., Rubin, H. and Schug, J. (2001) Hybrid modeling and simulation of biomolecular networks. *Proc. of Hybrid Systems: Computation and Control*, LNCS 2034, Springer, 19-32.

[2] Barbuti, R., Maggiolo-Schettini, A. and Milazzo, P. (2007) Extending the calculus of looping sequences to model protein interaction at the domain level. *Proc. of International Symposium on Bioinformatics Research and Applications (ISBRA'07)*, LNBI 4463, Springer, 638-649.

[3] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P. and Troina, A. (2006) A calculus of looping sequences for modelling microbiological systems. *Fund. Inform.*, **72**, 21-35.

[4] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P. and Troina, A. (2006) Bisimulation congruences in the calculus of looping sequences. *Proc. of International Colloquium on Theoretical Aspects of Computing (ICTAC'06)*, LNCS 4281, Springer, 93-107.

[5] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., and Troina, A. (2008) Bisimulations in calculi modelling membranes. *Formal Aspects of Computing*, to appear.

[6] Cardelli, L. (2005) Brane calculi. Interactions of biological membranes. *Proc. of Comput. Methods in Systems Biology (CMSB'04)*, LNCS 3082, Springer, 257-280.

[7] Danos, V. and Laneve, C. (2004) Formal molecular biology. *Theor. Comput. Sci.*, **325**, 69-110.

[8] McIntosh, T., Kulkarni, K., and Simon, S. (1999) Membrane fusion promoters and inhibitors have contrasting effects on lipid bilayer structure and undulations. *Biophysical Journal* **76**, 2090-2098.

[9] Matsuno, H., Doi, A., Nagasaki, M. and Miyano, S. (2000) Hybrid Petri net representation of gene regulatory network. *Prooceedings of Pacific Symposium on Biocomputing*, World Scientific Press, 341-352.

[10] Milazzo, P. (2007) *Qualitative and quantitative formal modeling of biological systems*. Ph.D. Thesis, University of Pisa.

[11] Păun, G. (2002) *Membrane computing. An introduction.* Springer, 2002.

[12] Regev, A., Panina, E. M., Silverman, W., Cardelli, L. and Shapiro, E. (2004) BioAmbients: an abstraction for biological compartments. *Theor. Comput. Sci.*, **325**, 141-167.

[13] Regev, A. and Shapiro, E. (2002) Cells as computation. *Nature*, **419**, 343.

[14] Regev, A. and Shapiro, E. (2004) The $\pi$-calculus as an abstraction for biomolecular systems. *Modelling in Molecular Biology*, Natural Computing Series, Springer, 219-266.

[15] Wells, J. (2002). The Essence of Principal Typings. *Proc. of 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, LNCS 2380, Springer, 913-925.