A Type System for Required/Excluded Elements in CLS*

Mariangiola Dezani-Ciancaglini
Dipartimento di Informatica, Università di Torino
dezani@di.unito.it

Paola Giannini
Dipartimento di Informatica, Università del Piemonte Orientale
giannini@mfn.unipmn.it

Angelo Troina
Dipartimento di Informatica, Università di Torino
troina@di.unito.it

The calculus of looping sequences is a formalism for describing the evolution of biological systems by means of term rewriting rules. We enrich this calculus with a type discipline to guarantee the soundness of reduction rules with respect to some biological properties deriving from the requirement of certain elements, and the repellency of others. As an example, we model a toy system where the repellency of a certain element is captured by our type system and forbids another element to exit a compartment.

1 Introduction

While the approach of biologists to describe biological systems by mathematical means, allows them to reason on the behaviour of the described systems and to perform quantitative simulations, such modelling starts becoming more difficult both in specification and in analysis when the complexity of the described systems increases. This has become one of the main motivations for the application of Computer Science formalisms to the description of biological systems [15]. Other motivations can be found in the fact that the use of formal means of Computer Science permits the application of analysis methods that are practically unknown to biologists, such as static analysis and model checking.

Many formalisms have either been applied to or have been inspired from biological systems. The most notable are automata-based models [1, 11], rewrite systems [9, 13], and process calculi [15, 16, 14, 8]. Models based on automata have the great advantage of allowing the direct use of many verification tools such as model checkers. On the other side, models based on rewrite systems usually allow describing biological systems with a notation that can be easily understood by biologists. However, automata-like 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.

In [5, 6, 12], Milazzo et al. 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 [6, 7], 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 chemistry, hydrophobicity is the physical property of a molecule (known as a hydrophobe) that is repelled from a mass of water. Hydrophobic molecules tend to be non–polar and thus prefer other neutral molecules and non–polar solvents. Hydrophobic molecules in water often cluster together forming micelles. From the other perspective, water on hydrophobic surfaces will exhibit a high contact angle (thus

*This work was partly funded by the project BioBIT of the Regione Piemonte.

causing, for example, the familiar dew drops on a hydrophobic leaf surface). Examples of hydrophobic molecules include the alkanes, oils, fats, and greasy substances in general. Hydrophobic materials are used for oil removal from water, the management of oil spills, and chemical separation processes to remove non-polar from polar compounds. Hydrophobicity is just an example of repellency in Biochemistry. Other well known examples may be found on the behaviour of anions and cations, or at a different level of abstraction, in the behaviour of the rh antigen for the different blood types.

As a counterpart, there may be elements, in nature, which always require the presence of other elements (it is difficult to find a lonely atom of oxygen, they always appear in the pair O_2).

In this paper we bring these aspects at their maximum limit, and, by abstracting away all the phenomena which give rise/arise to/from repellency (and its counterpart), we assume that for each kind of element of our reality we are able to fix a set of elements which are required by the element for its existence and a set of elements whose presence is forbidden by the element.

Thus, we enrich CLS with a type discipline which allows to guarantee the soundness of reduction rules with respect to some relevant properties of biological systems deriving from the required and excluded kinds of elements. The key technical tool we use is to 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 [3, 2] are the only papers which study type disciplines for CLS. We generalise the proposal in [2] by focusing on the type disciplines for Present/Required/Excluded elements.

2 The Calculus of Looping Sequences

In this section we recall the Calculus of Looping Sequences (CLS). CLS is essentially based on term rewriting, hence a CLS model consists of a term and a set of rewrite rules. The term is intended to represent the structure of the modelled system, and the rewrite rules to represent the events that may cause the system to evolve.

We start with defining the syntax of terms. We assume a possibly infinite alphabet $\mathscr{E}$ of symbols ranged over by $a, b, c, \ldots$.

Definition 2.1 (Terms) Terms T *and* sequences S *of* CLS *are given by the following grammar:*

$$\begin{array}{lcl} T & ::= & S \mid (S)^L \rfloor T \mid T \mid T \\ S & ::= & \varepsilon \mid a \mid S \cdot S \end{array}$$

where a is a generic element of $\mathscr{E}$, and ε represents the empty sequence. We denote with $\mathscr{T}$ the infinite set of terms, and with $\mathscr{S}$ the infinite set of sequences.

In CLS we have a sequencing operator $\_\cdot\_$, a looping operator $(\_)^L$, a parallel composition operator $\_\mid\_$ and a containment operator $\_\rfloor\_$. Sequencing can be used to concatenate elements of the alphabet $\mathscr{E}$. The empty sequence ε 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 definition, looping and containment are always applied together, hence we can consider them as a single binary operator $(\_)^L \rfloor \_$ which 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

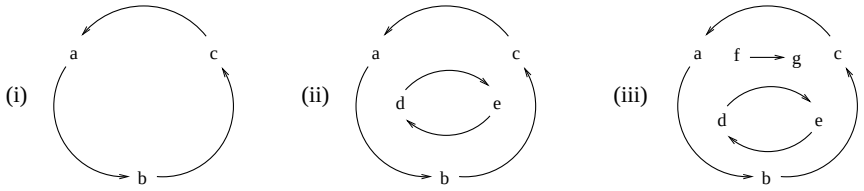

Figure 1: (i) represents $(a \cdot b \cdot c)^L$; (ii) represents $(a \cdot b \cdot c)^L \rfloor (d \cdot e)^L$; (iii) represents $(a \cdot b \cdot c)^L \rfloor ((d \cdot e)^L \mid f \cdot g)$.

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 sequence 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. Membranes are closed surfaces, often interspersed with proteins, which may contain something. 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 the content of the membrane. Other macro–molecules can be modelled as single alphabet symbols, or as short sequences. Finally, juxtaposition of entities can be described by the parallel composition of their representations.

Brackets can be used to indicate the order of application of the operators, and we assume $(\_)^L \rfloor \_$ to have precedence over $\_ \mid \_$. In Figure 1 we show some examples of CLS terms and their visual representation, using $(S)^L$ as a short-cut for $(S)^L \rfloor \varepsilon$.

In CLS we may have syntactically different terms representing the same structure. We introduce a structural congruence relation to identify such terms.

Definition 2.2 (Structural Congruence) *The structural congruence relations $\equiv_S$ and $\equiv_T$ are the least congruence relations on sequences and on terms, respectively, satisfying the following rules:*

$$S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3 \qquad S \cdot \varepsilon \equiv_S \varepsilon \cdot S \equiv_S S$$
$$S_1 \equiv_S S_2 \ \textit{implies}\ S_1 \equiv_T S_2 \ \textit{and}\ (S_1)^L \rfloor T \equiv_T (S_2)^L \rfloor T$$
$$T_1 \mid T_2 \equiv_T T_2 \mid T_1 \qquad T_1 \mid (T_2 \mid T_3) \equiv_T (T_1 \mid T_2) \mid T_3 \qquad T \mid \varepsilon \equiv_T T$$
$$(\varepsilon)^L \rfloor \varepsilon \equiv_T \varepsilon \qquad (S_1 \cdot S_2)^L \rfloor T \equiv_T (S_2 \cdot S_1)^L \rfloor T$$

Rules of the structural congruence state the associativity of $\cdot$ and $\mid$, the commutativity of the latter and the neutral role of ε. Moreover, axiom $(S_1 \cdot S_2)^L \rfloor T \equiv_T (S_2 \cdot S_1)^L \rfloor T$ says that looping sequences can rotate. In the following, for simplicity, we will use $\equiv$ in place of $\equiv_T$.

Rewrite rules will be defined essentially as pairs of terms, with the first term describing the portion of the system in which the event modelled by the rule may occur, and the second term describing 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 of these 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 $\mathscr{TV}$ ranged over by $X, Y, Z, \ldots$, a set of sequence variables $\mathscr{SV}$ ranged over by $\tilde{x}, \tilde{y}, \tilde{z}, \ldots$, and a set of element variables $\mathscr{X}$ ranged over by $x, y, z, \ldots$. All these sets

are possibly infinite and pairwise disjoint. We denote by $\mathscr{V}$ the set of all variables, $\mathscr{V} = \mathscr{TV} \cup \mathscr{SV} \cup \mathscr{X}$, and with ρ a generic variable of $\mathscr{V}$. Hence, a pattern is a term that may include variables.

Definition 2.3 (Patterns) Patterns *P and* sequence patterns *SP of* CLS *are given by the following grammar:*

$$\begin{array}{lll} P & ::= & SP \mid (SP)^L \rfloor P \mid P \mid P \mid X \\ SP & ::= & \varepsilon \mid a \mid SP \cdot SP \mid \widetilde{x} \mid x \end{array}$$

where a is a generic element of $\mathscr{E}$, and $X, \widetilde{x}$ and x are generic elements of $\mathscr{TV}, \mathscr{SV}$ and $\mathscr{X}$, respectively. We denote with $\mathscr{P}$ the infinite set of patterns.

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

Definition 2.4 (Rewrite Rules) *A rewrite rule is a pair of patterns (P_1, P_2), denoted with $P_1 \mapsto P_2$, where $P_1, P_2 \in \mathscr{P}$, $P_1 \not\equiv \varepsilon$ 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 σ, 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.

We define the semantics of CLS by resorting to the notion of contexts.

Definition 2.5 (Contexts) Contexts *C are defined as:*

$$C ::= \Box \mid C \mid T \mid T \mid C \mid (S)^L \rfloor C$$

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

By definition, every context contains a single hole $\Box$. Let us assume $C, C' \in \mathscr{C}$. With $C[T]$ we denote the term obtained by replacing $\Box$ with T in C; with $C[C']$ we denote context composition, whose result is the context obtained by replacing $\Box$ with C' in C. The structural equivalence is extended to contexts in the natural way (i.e. by considering $\Box$ as a new and unique symbol of the alphabet $\mathscr{E}$).

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 $a \mapsto b \mid c$) applied to an element of a sequence (e.g., $m \cdot a \cdot m$) would result into a syntactically incorrect term (in this case $m \cdot (b \mid c) \cdot m$). To modify a sequence, a pattern representing the whole sequence must appear in the rule. For example, rule $a \cdot \widetilde{x} \mapsto a \mid \widetilde{x}$ can be applied to any sequence starting with element a, and, hence, the term $a \cdot b$ can be rewritten as $a \mid b$, and the term $a \cdot b \cdot c$ can be rewritten as $a \mid b \cdot c$.

The semantics of CLS is defined as follows.

Definition 2.6 (Semantics) *Given a finite set of rewrite rules $\mathscr{R}$, the* semantics *of CLS is the least relation closed with respect to $\equiv$ and satisfying the following rule:*

$$\frac{P_1 \mapsto P_2 \in \mathscr{R} \quad P_1\sigma \not\equiv \varepsilon \quad \sigma \in \Sigma \quad C \in \mathscr{C}}{C[P_1\sigma] \to C[P_2\sigma]}$$

As usual we denote with $\rightarrow^*$ the reflexive and transitive closure of $\rightarrow$.

Given a set of rewrite rules $\mathscr{R}$, the behaviour of a term T is the tree of terms to which T may reduce. Thus, a *model* in CLS is given by a term describing the initial state of the system and by a set of rewrite rules describing all the events that may occur.

3 A Type Discipline for Required and Excluded Elements

We classify elements in $\mathscr{E}$ with *basic types*. Intuitively, given a molecule represented by an element in $\mathscr{E}$, we associate to it a type $\mathtt{t}$ which specifies the kind of the molecule. We assume a fixed typing Γ for the elements in $\mathscr{E}$.

For each basic type $\mathtt{t}$ we assume to have a pair of sets of types $(\mathtt{R_t}, \mathtt{E_t})$, where $\mathtt{t} \notin \mathtt{R_t} \cup \mathtt{E_t}$ and $\mathtt{R_t} \cap \mathtt{E_t} = \emptyset$, saying that the presence of elements of type $\mathtt{t}$ requires the presence of elements whose type is in $\mathtt{R_t}$ and forbids the presence of elements whose type is in $\mathtt{E_t}$. We consider only *local* properties: elements influence each other if they are either in the same compartment or they contain each other.

The type system infers the set of types of the elements of terms, checking that the constraints imposed by the required and excluded types are satisfied. Types are pairs $(\mathtt{P},\mathtt{R})$: where $\mathtt{P}$ is the set of types of *present* elements (at the top level of a pattern), $\mathtt{R}$ is the set of types of *required* elements (that should still be added to the term to represent a *correct* system). The set of *excluded* elements is implicitly given by $E_\mathtt{P} = \bigcup_{\mathtt{t}\in\mathtt{P}} \mathtt{E_t}$.

Types are well formed, and pair of types are compatible, if their constraints on required and excluded elements are not contradictory; compatible types can be combined.

Definition 3.1 (Auxiliary definitions)

- *A type* $(\mathtt{P},\mathtt{R})$ *is* well formed *if* $\mathtt{P} \cap E_\mathtt{P} = \mathtt{P} \cap \mathtt{R} = \mathtt{R} \cap E_\mathtt{P} = \emptyset$.
- *Well formed types* $(\mathtt{P},\mathtt{R})$ and $(\mathtt{P}',\mathtt{R}')$ are compatible *(written* $(\mathtt{P},\mathtt{R}) \bowtie (\mathtt{P}',\mathtt{R}')$*) if*
  - $E_\mathtt{P} \cap \mathtt{P}' = E_\mathtt{P} \cap \mathtt{R}' = \emptyset$*, and*
  - $E_{\mathtt{P}'} \cap \mathtt{P} = E_{\mathtt{P}'} \cap \mathtt{R} = \emptyset$.
- *Given two compatible types* $(\mathtt{P},\mathtt{R})$ *and* $(\mathtt{P}',\mathtt{R}')$ *we define their* conjunction $(\mathtt{P},\mathtt{R}) \sqcup (\mathtt{P}',\mathtt{R}')$ *by*

$$(\mathtt{P},\mathtt{R}) \sqcup (\mathtt{P}',\mathtt{R}') = (\mathtt{P} \cup \mathtt{P}', (\mathtt{R} \cup \mathtt{R}') \setminus (\mathtt{P} \cup \mathtt{P}')).$$

Basis are defined by:

$$\Delta \ ::= \ \emptyset \quad \mid \quad \Delta, x : (\{\mathtt{t}\}, \mathtt{R_t}) \quad \mid \quad \Delta, \eta : (\mathtt{P},\mathtt{R})$$

where η denotes a sequence or term variable. A basis Δ is *well formed* if all types in the basis are well formed.

We check the safety of terms, sequences and more generally patterns using the typing rules of Figure 2. It is easy to verify that if we start from well-formed basis, then in a derivation we produce only well-formed basis and well-formed types. Note that terms and sequences are typable from the empty context. All the rules are obvious except for the last one which types looping sequences. In this rule we can put a pattern *P* inside a looping sequence *SP* only if all the types required from *P* are provided by *SP*. This is because if *P* gets inside a compartment (represented by the looping sequence) it cannot interact any more with the environment.

Given a context we define the possible types of terms that may fill the hole in the context.

Definition 3.2 (Typed Holes) *Given a context C, and a well-formed type* $(\mathtt{P},\mathtt{R})$*, the type* $(\mathtt{P},\mathtt{R})$ *is* OK *for the context C if* $X : (\mathtt{P},\mathtt{R}) \vdash C[X] : (\mathtt{P}',\emptyset)$ *for some* $\mathtt{P}'$.

$$\Delta, \rho : (\mathtt{P},\mathtt{R}) \vdash \rho : (\mathtt{P},\mathtt{R}) \qquad \Delta \vdash \varepsilon : (\emptyset,\emptyset) \qquad \frac{a : \mathtt{t} \in \Gamma}{\Delta \vdash a : (\{\mathtt{t}\}, \mathtt{R}_\mathtt{t})}$$

$$\frac{\Delta \vdash SP : (\mathtt{P},\mathtt{R}) \quad \Delta \vdash SP' : (\mathtt{P}',\mathtt{R}') \quad (\mathtt{P},\mathtt{R}) \bowtie (\mathtt{P}',\mathtt{R}')}{\Delta \vdash SP \cdot SP' : (\mathtt{P},\mathtt{R}) \sqcup (\mathtt{P}',\mathtt{R}')}$$

$$\frac{\Delta \vdash P : (\mathtt{P},\mathtt{R}) \quad \Delta \vdash P' : (\mathtt{P}',\mathtt{R}') \quad (\mathtt{P},\mathtt{R}) \bowtie (\mathtt{P}',\mathtt{R}')}{\Delta \vdash P \,|\, P' : (\mathtt{P},\mathtt{R}) \sqcup (\mathtt{P}',\mathtt{R}')}$$

$$\frac{\Delta \vdash SP : (\mathtt{P},\mathtt{R}) \quad \Delta \vdash P : (\mathtt{P}',\mathtt{R}') \quad (\mathtt{P},\mathtt{R}) \bowtie (\mathtt{P}',\mathtt{R}') \text{ and } \mathtt{R}' \subseteq \mathtt{P}}{\Delta \vdash (SP)^L \rfloor P : (\mathtt{P}, \mathtt{R} \setminus \mathtt{P}')}$$

Figure 2: Typing rules for Present/Required/Excluded Elements

The above notion guarantees that filling a context with a term we obtain a correct system (whose type is well formed and whose requirements are completely satisfied). It is to this kind of terms that we are interested in applying reduction rules.

Note that there may be more than one type $(\mathtt{P},\mathtt{R})$ such that $(\mathtt{P},\mathtt{R})$ is OK for the context C.

We can classify reduction rules according to the types we can derive for the right hand sides of the rules.

Definition 3.3 (Δ-$(\mathtt{P},\mathtt{R})$-Reduction Rules) *A rule $P_1 \mapsto P_2$ is a* Δ-$(\mathtt{P},\mathtt{R})$-reduction rule *if* $\Delta \vdash P_2 : (\mathtt{P},\mathtt{R})$.

An instantiation σ *agrees* with a basis Δ (notation $\sigma \in \Sigma_\Delta$) if $\rho : (\mathtt{P},\mathtt{R}) \in \Delta$ implies $\vdash \sigma(\rho) : (\mathtt{P},\mathtt{R})$.

We can safely apply a rule to a typed term only if the instances of the pattern on the right hand side of the rule has a type that is *OK* for the context. More formally:

Definition 3.4 (Typed Semantics) *Given a finite set of rewrite rules $\mathscr{R}$, the* typed semantics *of* CLS *is the least relation closed with respect to $\equiv$ and satisfying the following rule:*

$$\frac{\begin{array}{c} P_1 \mapsto P_2 \in \mathscr{R} \text{ is a } \Delta\text{-}(\mathtt{P},\mathtt{R})\text{-reduction rule} \qquad P_1\sigma \not\equiv \varepsilon \\ \sigma \in \Sigma_\Delta \qquad C \in \mathscr{C} \qquad (\mathtt{P},\mathtt{R}) \text{ is OK for } C \end{array}}{C[P_1\sigma] \Longrightarrow C[P_2\sigma]}$$

As expected, reduction preserves typing, in the sense that the obtained term is still typable, but the new type can have a different set of present elements, while the set of required elements is always empty. This choice makes possible typing creation and degradation of elements.

Theorem 3.5 *If $\vdash T : (\mathtt{P},\emptyset)$ and $T \Longrightarrow T'$, then $\vdash T' : (\mathtt{P}',\emptyset)$ for some $\mathtt{P}'$.*

We can infer the OK relation between types and contexts and which rules are Δ-$(\mathtt{P},\mathtt{R})$-reduction rules by using the machinery of principal typing [17]. In this way we can decide the applicability of the reduction rules for the typed semantics. This is the content of the remaining part of this section.

We convene that for each variable $x \in \mathscr{X}$ there is an *e-type variable* φ_x ranging over basic types, and for each variable $\eta \in \mathscr{TV} \cup \mathscr{SV}$ there are two variables ϕ_η, ψ_η (called *p-type variable* and *r-type variable*) ranging over sets of basic types. Moreover we convene that Φ ranges over formal unions and differences of sets of basic types, e-type variables and p-type variables, and Ψ ranges over formal unions and differences of sets of basic types and r-type variables.

$$\vdash \varepsilon : \emptyset;(\emptyset,\emptyset);\emptyset \qquad \frac{a:\mathtt{t}\in\Gamma}{\vdash a:\emptyset;(\mathtt{t},\mathtt{R}_\mathtt{t});\emptyset} \qquad \vdash x:\{x:(\varphi_x,\Psi)\};(\varphi_x,\Psi);\{\Psi=\mathtt{R}_{\varphi_x}\}$$

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

$$\frac{\vdash SP:\Theta;(\Phi,\Psi);\Xi \qquad \vdash SP':\Theta';(\Phi',\Psi');\Xi'}{\vdash SP\cdot SP':\Theta\cup\Theta';(\Phi,\Psi)\sqcup(\Phi',\Psi');\Xi\cup\Xi'\cup\{(\Phi,\Psi)\bowtie(\Phi',\Psi')\}}$$

$$\frac{\vdash P:\Theta;(\Phi,\Psi);\Xi \qquad \vdash P':\Theta';(\Phi',\Psi');\Xi'}{\vdash P\mid P':\Theta\cup\Theta';(\Phi,\Psi)\sqcup(\Phi',\Psi');\Xi\cup\Xi'\cup\{(\Phi,\Psi)\bowtie(\Phi',\Psi')\}}$$

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

Figure 3: Inference Rules for Principal Typing

A *basis scheme* Θ is a mapping from atomic variables to their e-type variables, and from sequence and term variables to pairs of their p-type variables and r-type variables:

$$\Theta \ ::= \ \emptyset \ \mid \ \Theta,x:\varphi_x \ \mid \ \Theta,\eta:(\phi_\eta,\psi_\eta).$$

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

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

where Θ is the *principal basis* in which P is well formed, (Φ,Ψ) is the *principal type* of P, and Ξ is the set of constraints that should be satisfied. 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, p-type variables and r-type variables to sets of basic types. A type mapping m *satisfies* a set of constraints Ξ if all constraints in $\mathsf{m}(\Xi)$ are satisfied.

Theorem 3.6 (Soundness of Type Inference) *If* $\vdash P:\Theta;(\Phi,\Psi);\Xi$ *and* m *is a type mapping which satisfies* Ξ*, then* $\mathsf{m}(\Theta)\vdash P:(\mathsf{m}(\Phi),\mathsf{m}(\Psi))$.

Theorem 3.7 (Completeness of Type Inference) *If* $\Delta\vdash P:(\mathtt{P},\mathtt{R})$*, then* $\vdash P:\Theta;(\Phi,\Psi);\Xi$ *for some* Θ*,* (Φ,Ψ)*,* Ξ *and there is a type mapping* m *that satisfies* Ξ *and such that* $\Delta\supseteq\mathsf{m}(\Theta)$*,* $\mathtt{P}=\mathsf{m}(\Phi)$*,* $\mathtt{R}=\mathsf{m}(\Psi)$.

We put now our inference rules at work in order to decide applicability of typed reduction rules. We first characterize by means of principal typing the OK relation and the classification of reduction rules.

Notably for deciding the OK relation it is not necessary to consider the whole context, but only the part of the context which influences the typing of the hole. The key observation is that the typing of a term inside two nested looping sequences does not depend on the typing of the terms outside the outermost looping sequence. We call *core of the context* that part. More formally:

Definition 3.8 *The* core of the context C *(notation* $\mathsf{core}(C)$*) is defined by:*

- $\mathsf{core}(C)=C$ *if* $C\equiv\Box\mid T_1$ *or* $C\equiv(S_1)^L\rfloor(\Box\mid T_1)\mid T_2$*;*
- $\mathsf{core}(C_1[C_2])=C_2$ *if* $C_2\equiv(S_2)^L\rfloor((S_1)^L\rfloor(\Box\mid T_1)\mid T_2)$.

Remark that core is always unambiguously defined, since every context can be split in an unique way into two contexts satisfying the given conditions.

Lemma 3.9 (OK Relation) *Let the context C be such that $\vdash C[T] : (\mathtt{P}_0, \emptyset)$ for some $T, \mathtt{P}_0$. A type $(\mathtt{P}, \mathtt{R})$ is OK for C if and only if the type mapping m defined by*

1. $\mathsf{m}(\phi_X) = \mathtt{P}$,
2. $\mathsf{m}(\psi_X) = \mathtt{R}$,

satisfies the set of constraints

$$\Xi \cup \{\Psi = \emptyset \mid \text{ if } \phi_X \text{ or } \psi_X \text{ occurs in } \Psi\},$$

where $\vdash \mathsf{core}(C)[X] : \{X : (\phi_X, \psi_X)\}; (\Phi, \Psi); \Xi$.

It is easy to check that if $\mathsf{core}(C) \equiv (S_2)^L \rfloor ((S_1)^L \rfloor (\Box \mid T_1) \mid T_2)$, and $\vdash T_1 : (\mathtt{P}_1, \mathtt{R}_1)$, $\vdash S_1 : (\mathtt{P}'_1, \mathtt{R}'_1)$, $\vdash T_2 : (\mathtt{P}_2, \mathtt{R}_2)$, $\vdash S_2 : (\mathtt{P}'_2, \mathtt{R}'_2)$, then we get the following six constraints to verify:

- $(\phi_X, \psi_X) \bowtie (\mathtt{P}_1, \mathtt{R}_1)$
- $(\mathtt{P}'_1, \mathtt{R}'_1) \bowtie ((\phi_X, \psi_X) \sqcup (\mathtt{P}_1, \mathtt{R}_1))$
- $((\psi_X \cup \mathtt{R}_1) \setminus (\phi_X \cup \mathtt{P}_1)) \subseteq \mathtt{P}'_1$
- $(\mathtt{P}'_1, \mathtt{R}'_1 \setminus (\phi_X \cup \mathtt{P}_1)) \bowtie (\mathtt{P}_2, \mathtt{R}_2)$
- $(\mathtt{P}'_2, \mathtt{R}'_2) \bowtie ((\mathtt{P}'_1, \mathtt{R}'_1 \setminus (\phi_X \cup \mathtt{P}_1)) \sqcup (\mathtt{P}_2, \mathtt{R}_2))$
- $(((\mathtt{R}'_1 \setminus (\phi_X \cup \mathtt{P}_1)) \cup \mathtt{R}_2) \setminus (\mathtt{P}'_1 \cup \mathtt{P}_2)) \subseteq \mathtt{P}'_2$.

The set of constraints simplifies when the core context is shorter.

Lemma 3.10 (Classification of Reduction Rules) *A rule $P_1 \mapsto P_2$ is a Δ-$(\mathtt{P}, \mathtt{R})$-reduction rule if and only if the type mapping m defined by*

1. $\mathsf{m}(\varphi_x) = \mathtt{t}$ *if* $\Delta(x) = (\{\mathtt{t}\}, \mathtt{R}_\mathtt{t})$,
2. $\mathsf{m}(\phi_\eta) = \mathtt{P}'$ *if* $\Delta(\eta) = (\mathtt{P}', \mathtt{R}')$,
3. $\mathsf{m}(\psi_\eta) = \mathtt{R}'$ *if* $\Delta(\eta) = (\mathtt{P}', \mathtt{R}')$,

satisfies the set of constraints $\Xi \cup \{\Phi = \mathtt{P}, \Psi = \mathtt{R}\}$, where $\vdash P_2 : \Theta; (\Phi, \Psi); \Xi$.

The above two lemmas, whose proof from the given definitions is easy, imply the following theorem which gives the desired result.

Theorem 3.11 (Applicability of Reduction Rules) *Let*

$$\vdash P_2 : \Theta; (\Phi, \Psi); \Xi \text{ and } \vdash \mathsf{core}(C)[X] : \{X : (\phi_X, \psi_X)\}; (\Phi', \Psi'); \Xi'.$$

Then the rule $P_1 \mapsto P_2$ can be applied to the term $C[P_1\sigma]$ such that $\vdash C[P_1\sigma] : (\mathtt{P}, \emptyset)$ for some $\mathtt{P}$ if and only if the type mapping m defined by

1. $\mathsf{m}(\varphi_x) = \mathtt{t}$ *if* $\sigma(x) : \mathtt{t} \in \Gamma$,
2. $\mathsf{m}(\phi_\eta) = \mathtt{P}'$ *if* $\vdash \sigma(\eta) : (\mathtt{P}', \mathtt{R}')$,
3. $\mathsf{m}(\psi_\eta) = \mathtt{R}'$ *if* $\vdash \sigma(\eta) : (\mathtt{P}', \mathtt{R}')$,

satisfies the set of constraints $\Xi \cup \Xi' \cup \{\Phi = \phi_X, \Psi = \psi_X\} \cup \{\Psi' = \emptyset \mid \text{ if } \phi_X \text{ or } \psi_X \text{ occurs in } \Psi'\}$.

Note that - after fixing the reduction rules - the sets of constraints for typing the r.h.s. of these rules can be evaluated once for all. Instead, the sets of constraints for typing the core contexts need to be evaluated at every application of a reduction rule. Luckily these sets of constraints includes at most six constraints. The mapping m is immediate from the derivation of a type for $P_1\sigma$. Finally, the checking that m satisfies a set of constraints requires only some substitutions.

4 Examples

We start showing the properties of our type system by modelling an example of two molecules repelling each other. As we have seen, one might model repellency in our framework via the set $\mathtt{E}_\mathtt{t}$.

Namely, if molecule a, of basic type $\mathtt{t}$, is a repellent for molecule b, of basic type $\mathtt{t}'$ (and viceversa), we will have that $\mathtt{E}_\mathtt{t} = \{\mathtt{t}'\}$ and $\mathtt{E}_{\mathtt{t}'} = \{\mathtt{t}\}$. Note that this does not mean that a and b cannot be present in the same term, actually they should just be contained in two different compartments. In fact, the term

$$T = a \,|\, (m)^L \rfloor\, b$$

with m of basic type $\mathtt{t}''$, where $\mathtt{E}_{\mathtt{t}''} = \emptyset$ and $\mathtt{R}_\mathtt{t} = \mathtt{R}_{\mathtt{t}'} = \mathtt{R}_{\mathtt{t}''} = \emptyset$, is typed by the pair $(\{\mathtt{t}, \mathtt{t}''\}, \emptyset)$ by the following derivation, where $\Gamma = \{a : \mathtt{t}, b : \mathtt{t}', m : \mathtt{t}''\}$:

$$\dfrac{\dfrac{a : \mathtt{t} \in \Gamma}{\vdash a : (\{\mathtt{t}\}, \emptyset)} \qquad \dfrac{\dfrac{m : \mathtt{t}'' \in \Gamma}{\vdash m : (\{\mathtt{t}''\}, \emptyset)} \qquad \dfrac{b : \mathtt{t}' \in \Gamma}{\vdash b : (\{\mathtt{t}'\}, \emptyset)} \quad (\{\mathtt{t}''\}, \emptyset) \bowtie (\{\mathtt{t}'\}, \emptyset)}{\vdash (m)^L \rfloor\, b : (\{\mathtt{t}''\}, \emptyset)} \quad (\{\mathtt{t}''\}, \emptyset) \bowtie (\{\mathtt{t}\}, \emptyset)}{\vdash a \,|\, (m)^L \rfloor\, b : (\{\mathtt{t}, \mathtt{t}''\}, \emptyset)}$$

As we can see the term is well typed, since a and b are in two different compartments. The element m could also be replaced by any sequence whose (well formed) type is $(\mathtt{P}, \mathtt{R})$ where $(\mathtt{P}, \mathtt{R}) \bowtie (\{\mathtt{t}'\}, \emptyset)$ and $(\mathtt{P}, \mathtt{R}) \bowtie (\{\mathtt{t}\}, \emptyset)$, that is $\mathtt{P} \cap \{\mathtt{t}, \mathtt{t}'\} = \emptyset$ and $\mathtt{R} \cap \{\mathtt{t}, \mathtt{t}'\} = \emptyset$. Moreover, if the term is at top level, then $\mathtt{R}$ should be $\emptyset$.

Consider now the rule

$$R_1 : (\widetilde{x})^L \rfloor\, (X \,|\, b) \mapsto b \,|\, (\widetilde{x})^L \rfloor\, X$$

which moves the element b outside the compartment. Such a rule could not be executed in T cause it will result in the term $a \,|\, b \,|\, (m)^L \rfloor\, \varepsilon$, which could not be typed since $(\{\mathtt{t}, \mathtt{t}', \mathtt{t}''\}, \emptyset)$ is not well formed. The rule could not be fired cause, following Definition 3.4, $P_1 = (\widetilde{x})^L \rfloor\, (X \,|\, b)$, $P_2 = b \,|\, (\widetilde{x})^L \rfloor\, X$, $\sigma(\widetilde{x}) = m$, $\sigma(X) = \varepsilon$ and $C = a \,|\, \Box$. Now, while $C[P_1\sigma]$ is well typed, the same does not hold for $C[P_2\sigma]$.

This does not mean the rule R_1 could never be fired. In fact, if we consider the term

$$T' = a \,|\, (m)^L \rfloor\, b \,|\, (m)^L \rfloor\, \varepsilon$$

and we add the rule

$$R_2 : a \,|\, (\widetilde{x})^L \rfloor\, X \mapsto (\widetilde{x})^L \rfloor\, (a \,|\, X)$$

we still have that rule R_1 could not be fired on T' (same reasons as before), and we can neither execute the rule R_2 moving the a in the compartment containing b (in this case: $C = (m)^L \rfloor\, \varepsilon \,|\, \Box$, $P_2 = (\widetilde{x})^L \rfloor\, (a \,|\, X)$ with $\sigma(\widetilde{x}) = m$ and $\sigma(X) = b$, thus $P_2\sigma$ could not be typed). We can, however, apply rule R_2, moving a into the empty compartment. Namely, for $C = (m)^L \rfloor\, b \,|\, \Box$, $P_1 = a \,|\, (\widetilde{x})^L \rfloor\, X$, $P_2 = (\widetilde{x})^L \rfloor\, (a \,|\, X)$ with $\sigma(\widetilde{x}) = m$ and $\sigma(X) = \varepsilon$, we have $T' \equiv C[P_1\sigma]$ with type $(\{\mathtt{t}, \mathtt{t}''\}, \emptyset)$ and $T'' = C[P_2\sigma] = (m)^L \rfloor\, a \,|\, (m)^L \rfloor\, b$ with type $(\{\mathtt{t}''\}, \emptyset)$ - both $C[P_1\sigma]$ and $C[P_2\sigma]$ are well typed. Now, from T'' we can finally apply rule R_1, bringing b outside the compartment. The context of the rule application will be $C = (m)^L \rfloor\, a \,|\, \Box$ and the patterns $P_1 = (\widetilde{x})^L \rfloor\, (X \,|\, b)$, $P_2 = b \,|\, (\widetilde{x})^L \rfloor\, X$ where $\sigma(\widetilde{x}) = m$ and $\sigma(X) = \varepsilon$. Both $C[P_1\sigma]$ and $C[P_2\sigma]$ are well typed and the resulting term will be

$$T''' = b \,|\, (m)^L \rfloor\, a \,|\, (m)^L \rfloor\, \varepsilon.$$

We now show an application for the set of required elements in our typing system. The idea is to model the absorption of a given compound c by a cell. The absorption is promoted by a receptor r which should be present in the surface of the cell. We can model the effect of the absorption by using a different symbol (thus a different type) for the compound when it enters the cell, namely c'. The basic types of c, r and c' are, respectively, $\mathtt{t}$, $\mathtt{t}'$ and $\mathtt{t}''$. We also assume that $\mathtt{E}_{\mathtt{t}} = \mathtt{E}_{\mathtt{t}'} = \mathtt{E}_{\mathtt{t}''} = \mathtt{R}_{\mathtt{t}} = \mathtt{R}_{\mathtt{t}'} = \emptyset$. The requirement for $\mathtt{t}''$ (modelling the type of the compound inside a cell) should be, instead, the presence of the receptor on the membrane surface. We can model this condition with the set $\mathtt{R}_{\mathtt{t}''} = \{\mathtt{t}'\}$. Thus, with these basic types, we can model the rule for the absorption of the protein as:

$$R : c \,|\, (\widetilde{x})^L \rfloor X \mapsto (\widetilde{x})^L \rfloor (X \,|\, c')$$

without imposing explicitly the presence of the receptors on the patterns of the rule.

Actually, our type system guarantees that such a rule could not be applied to the term $c \,|\, (m)^L \rfloor \varepsilon$, while it could be applied to the term $c \,|\, (m \cdot r)^L \rfloor \varepsilon$.

In a sense, the role of the promoter (the receptor) is modelled intrinsically on the type of the compound brought inside the cell; its properties become transparent for the rewrite rules and controlled by the type system.

5 Conclusions

This paper is a first step toward the application of typing to the safety of system transformations which model biological phenomena. We focused on disciplines deriving by the requirement/exclusion of certain elements, and used the type system to describe how repellency could be modelled. We would like to underline that in nature it is not easy to find elements which completely exclude or require other elements. Our abstraction, however allows us to deal with a simple qualitative model, and to observe some basic properties of biological systems. A more detailed analysis, could also deal with quantities. In this case, typing is useful in modelling quantitative aspects of CLS semantics on the line of [4]. In particular, in [10], we show how types can be used to model repellency also by quantitative means, that is slowing down undesired interactions.

As a future work, 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.

An interesting application of this model may also abstract from biological phenomena. In a sense, the composition of a context C with a term T which satisfies C may represent the agreement on a *contract* between C and T. Namely, if C satisfies $(\mathtt{P},\mathtt{R})$ and T has type $(\mathtt{P},\mathtt{R})$, then T offers to C, via the elements in $\mathtt{P}$, everything that is *required* by C, viceversa, C has to satisfy T's request $\mathtt{R}$; modelling, in a sense, the fact that C and T mutually agree with each other.

Finally, modelling biological phenomena in CLS, especially for biologists, could be made more intuitive with the use of a graphical interface, that would check most of the syntactical details, so that the modeller could focus on the conceptual aspects of the formalization.

Acknowledgements. We thank the referees for their helpful comments. The final version of the paper improved due to their suggestions.

References

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

[2] Aman, B., Dezani-Ciancaglini, M., Troina, A. (2008) Type Disciplines for Analysing Biologically Relevant Properties. *Proc. of International Meeting on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'08)*, ENTCS 227, Elsevier, 97-111.

[3] 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.

[4] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tiberi, P.,Troina, A. (2008) Stochastic Calculus of Looping Sequences for the Modelling and Simulation of Cellular Pathways.*Transactions on Computational Systems Biology*, **vol. IX**, 86-113.

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

[6] 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.

[7] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., and Troina, A. (2008) Bisimulations in calculi modelling membranes. *Formal Aspects of Computing*, **20**(4-5), 351-377.

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

[9] Danos, V. and Laneve, C. (2004) Formal molecular biology. *Theoretical Computer Science*, **325**, 69-110.

[10] Dezani-Ciancaglini, M., Giannini, P. and Troina, A. (2009) A Type System for a Stochastic CLS. Unpublished, available from the authors home pages.

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

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

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

[14] Regev, A., Panina, E. M., Silverman, W., Cardelli, L. and Shapiro, E. (2004) BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science*, **325**, 141-167.

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

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

[17] 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.