Int J Software Informatics, Vol.XX, No.XX, XXXXXX 20XX, pp. 1–000 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
Tel: +86-10-62661040

Type Directed Semantics for the Calculus of Looping Sequences

Livio Bioglio[1], Mariangiola Dezani-Ciancaglini[1], Paola Giannini[2] and Angelo Troina[1]

[1]Dipartimento di Informatica, Università di Torino,

[2]Dipartimento di Informatica, Università del Piemonte Orientale

Abstract The calculus of looping sequences is a formalism for describing the evolution of biological systems by means of term rewriting rules. Here we enrich this calculus with a type discipline which preserves some biological properties deriving from the requirement of certain elements, and the repellency of others. In particular, the type system guarantees the soundness of the application of reduction rules with respect to the elements which are required (all requirements must be satisfied) and to the elements which are excluded (two elements which repel each other cannot occur in the same compartment).

As an example, we model the possible interactions (and compatibility) of different blood types with different antigens. The type system does not allow transfusion with incompatible blood types.

1 Introduction

While the approach of biologists to describe biological systems by mathematical means makes possible to reason on the behaviour of systems and to perform quantitative simulations, such modelling becomes more difficult both in specification and in analysis when the complexity of systems increases. This is one of the main motivations for the application of Computer Science formalisms to the description of biological systems [19]. 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 [2, 15], rewrite systems [11, 17], and process calculi [19, 20, 18, 10]. 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 describe biological systems with a notation that can be easily understood by biologists. However, automata-like models and rewrite systems are not compositional. Studying the behaviour of a system componentwise is in general ensured by process calculi, included those commonly used to describe biological systems.

In [7, 6, 16], 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

† This work is sponsored by the BioBITs Project (*Converging Technologies* 2007, area: Biotechnology-ICT), Regione Piemonte.

composition operator, and some semantic means, such as bisimulations [6, 8], which are common in process calculi. This permits to combine the notational simplicity 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 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 in 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 [13], we brought these aspects at their maximum limit, and, by abstracting away all the phenomena which give rise/arise to/from repellency (and its counterpart), we assumed 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. We enriched CLS with a type discipline which guarantees 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. This semantics is more liberal (in the sense of allowing the application of more rules) than the one of [3] in which only type preserving rules could be applied. However, the constraints that need to be verified, in order to apply a rule, are more complex than the ones of [3].

In this paper we show that the approach of [13] subsumes the one of [3] and propose a semantics that uses both. The advantage being that in the reduction process we first try to verify the constraints of [3], and in case they are not satisfied we pass to verify the ones of [13]. We also propose a type inference algorithm and show its soundness and completeness.

The required/excluded elements properties modelled in this paper assure, through type inference, that only compatible elements are combined together in the different environments of the biological system in consideration. Thus the type system intrinsically yields a notion of correct (well-behaving) system according to the expressed requirements.

While with our typed reductions from terms respecting the requests and repellency conditions we generate only terms with the same property, in reality also dangerous configurations can arise. We could adapt our typed semantics in order to allow the creation of untypable terms. In these cases, we might be interested in signalling that

some dangerous transition has occurred. We refer to the conclusion for a more technical discussion of this issue.

In the last few years there has been a growing interest in the use of type disciplines to enforce biological properties. In [3] a type system has been defined to ensure the well-formedness of links between protein sites within the Linked Calculus of Looping Sequences (see [4]). In [14] three type systems are defined for the Biochemical Abstract Machine, BIOCHAM (see [1]). The first one is used to infer the functions of proteins in a reaction model, the second one to infer activation and inhibition effects of proteins, and the last one to infer the topology of compartments. In [13] we have defined a type system for the Calculus of Looping Sequences (see [7]) to guarantee the soundness of reduction rules with respect to the requirement of certain elements, and the repellency of others.[1] In [12] we have proposed a type system for the Stochastic Calculus of Looping sequences (see [5]) that allows for a quantitative analysis and models how the presence of catalysers (or inhibitors) can modify the speed of reactions. Finally, in [9] we developed a type system to verify the excluded elements property for BioAmbients [18].

1.1 Summary

The remainder of the paper is organised as follows. In Section 2 we briefly introduce the calculus of looping sequences. In Section 3 we develop the type discipline for required and excluded elements and we embed it into the semantics of the calculus. In Section 4 we use the machinery of principal typing to infer the type of rewrite rules, and check their applicability. In Section 5 we apply our typing discipline to regulate the transfusion of different, possibly incompatible, blood types. Finally, we draw some conclusion.

2 The Calculus of Looping Sequences

The Calculus of Looping Sequences (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 $\mathcal{E}$ 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:

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

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

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

[1] The present paper is an improved and extended version of [13].

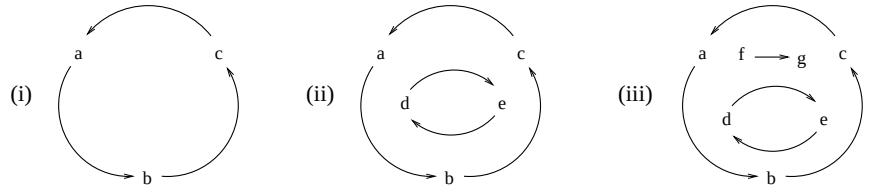

Fig. 1. (i) represents $(a \cdot b \cdot c)^L$; (ii) represents $(a \cdot b \cdot c)^L \rfloor (d \cdot e)^L$; (iii) represents $(a \cdot b \cdot c)^L \rfloor ((d \cdot e)^L \mid f \cdot g)$.

concatenate elements of the alphabet $\mathcal{E}$. 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.

We call *compartment* any parallel composition of one or more terms. Given a containment operator, $(S)^L \rfloor T$, its *looping sequence* is S and its *inner compartment* is the term T.

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 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 $(S)^L \rfloor \epsilon$.

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:

$$S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3 \qquad S \cdot \epsilon \equiv_S \epsilon \cdot S \equiv_S S$$
$$S_1 \equiv_S S_2 \text{ implies } S_1 \equiv_T S_2 \text{ and } (S_1)^L \rfloor T \equiv_T (S_2)^L \rfloor T$$
$$T_1 \,|\, T_2 \equiv_T T_2 \,|\, T_1 \qquad T_1 \,|\, (T_2 \,|\, T_3) \equiv_T (T_1 \,|\, T_2) \,|\, T_3 \qquad T \,|\, \epsilon \equiv_T T$$
$$(\epsilon)^L \rfloor \epsilon \equiv_T \epsilon \qquad (S_1 \cdot S_2)^L \rfloor T \equiv_T (S_2 \cdot S_1)^L \rfloor T$$

Rules of structural congruence state the associativity of $\cdot$ and $|$, the commutativity of the latter and the neutral role of ϵ. Moreover, the 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 $\equiv_T$.

We say that an element a *is present in a sequence* S if $S \equiv S' \cdot a \cdot S''$ for some S', S''. An element a *is present in a compartment* T if $T \equiv T' \,|\, T''$ for some T', T'' and either $T' = S$ or $T' = (S)^L \rfloor \_$ for some S and in both cases a is present in S.

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 $\mathcal{TV}$ ranged over by $X, Y, Z, \ldots$, a set of sequence variables $\mathcal{SV}$ 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 possibly infinite and pairwise disjoint. We denote by $\mathcal{V}$ the set of all variables, $\mathcal{V} = \mathcal{TV} \cup \mathcal{SV} \cup \mathcal{X}$, and with ρ a generic variable of $\mathcal{V}$. 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:

$$\begin{array}{lcl} P & ::= & SP \;\mid\; (SP)^L \rfloor P \;\mid\; P \,|\, P \;\mid\; X \\ SP & ::= & \epsilon \;\mid\; a \;\mid\; SP \cdot SP \;\mid\; \widetilde{x} \;\mid\; x \end{array}$$

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

We assume the structural congruence relation to be trivially extended to patterns. An *instantiation* is a partial function $\sigma : \mathcal{V} \to \mathcal{T}$. An instantiation must preserve the kind of variables, thus for $X \in \mathcal{TV}, \widetilde{x} \in \mathcal{SV}$ 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 Σ 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. Now we define rewrite rules.

Definition 2.4 (Rewrite Rules). A rewrite rule, $\Re$, is a pair of patterns, denoted with $P_1 \mapsto P_2$, where $P_1, P_2 \in \mathcal{P}$, $P_1 \not\equiv \epsilon$ and such that $Var(P_2) \subseteq Var(P_1)$.

Example 2.5. An example of rewrite rule is

$$(a \cdot \widetilde{x})^L \rfloor (b \,|\, Y) \mapsto b \,|\, (a \cdot \widetilde{x})^L \rfloor Y.$$

This rule says that the element b, alone, can exit from a looping sequence containing the element a.

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 2.6 (Contexts). *Contexts* C are defined as:

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

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

By definition, every context contains a single hole $\Box$. Let us assume $C, C' \in \mathcal{C}$. 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 $\mathcal{E}$).

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 $a \mapsto b \,|\, c$) 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.

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 (set of) rules:

$$\frac{\Re = P_1 \mapsto P_2 \in \mathcal{R} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma \quad C \in \mathcal{C}}{C[P_1\sigma] \rightarrow C[P_2\sigma]}$$

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

Given a set of rewrite rules $\mathcal{R}$, 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.

Example 2.8. Starting from the term

$$d \,|\, (a)^L \rfloor ((a \cdot c)^L \rfloor ((a \cdot b \cdot c)^L \rfloor (b)))$$

we can apply the rule in Example 2.5 three times, using the instantiations and contexts in Fig. 2, obtaining the behaviour

	$\widetilde{x}$	Y	C
(1)	$b \cdot c$	ϵ	$d \mid (a)^L \rfloor ((a \cdot c)^L \rfloor \Box)$
(2)	c	$(a \cdot b \cdot c)^L \rfloor \epsilon$	$d \mid (a)^L \rfloor \Box$
(3)	ϵ	$(a \cdot c)^L \rfloor ((a \cdot b \cdot c)^L \rfloor \epsilon))$	$d \mid \Box$

Fig. 2. Instantiations and Contexts of Example 2.8

$$\begin{aligned} d \mid (a)^L \rfloor ((a \cdot c)^L \rfloor ((a \cdot b \cdot c)^L \rfloor (b))) &\rightarrow d \mid (a)^L \rfloor ((a \cdot c)^L \rfloor (b \mid (a \cdot b \cdot c)^L \rfloor \epsilon)) \quad (*) \\ &\rightarrow d \mid (a)^L \rfloor (b \mid (a \cdot c)^L \rfloor ((a \cdot b \cdot c)^L \rfloor \epsilon)) \quad (**) \\ &\rightarrow b \mid d \mid (a)^L \rfloor ((a \cdot c)^L \rfloor ((a \cdot b \cdot c)^L \rfloor \epsilon)) \quad (***) \end{aligned}$$

2.1 Modelling Guidelines

CLS can be used to model biomolecular systems analogously to what is done, e.g, by Regev and Shapiro in [20] for the π-calculus. An abstraction is a mapping from a real-world domain to a mathematical domain, which may allow highlighting some essential properties of a system while ignoring other, complicated, ones. In [20], Regev and Shapiro show how to abstract biomolecular systems as concurrent computations by identifying the biomolecular entities and events of interest and by associating them with concepts of concurrent computations such as concurrent processes and communications.

The use of rewrite systems, such as CLS, to describe biological systems is founded on a different abstraction. Usually, entities (and their structures) are abstracted by terms of the rewrite system, and events by rewrite rules.

In order to describe cells, it is quite natural to consider molecular populations and membranes. Molecular populations are groups of molecules that are in the same compartment of the cell. As we have said before, molecules can be of many types: they could be classified as DNA and RNA strands, proteins, and other molecules.

DNA and RNA strands and proteins can be seen as non-elementary objects. DNA strands are composed by genes, RNA strands are composed by parts corresponding to the transcription of individual genes, and proteins are composed by parts having the role of interaction sites (or domains). Other molecules are considered as elementary objects, even if they are complexes.

Membranes are considered as elementary objects, in the sense that we do not describe them at the level of the lipids they are made of. The only interesting properties of a membrane are that it may have a content (hence, create a compartment) and that it may have molecules on its surface.

CLS is a very scalable formalism. On the one hand, depending on the level of detail one is interested in the analysis, an atomic element could range from the quark level (in a very low level analysis) to a species individual (in the study of population dynamics).[2] On the other hand, a looping sequence can be used to model cell compartmentalisation, or, from a macroscopic point of view, ecoregions bounded by

[2] Atoms, chemicals, molecules, protein domains, proteins, cells, etc. are other possible elements one could model, at different levels of abstraction, as CLS simple alphabet symbols.

Biomolecular Event	**Examples of CLS Rewrite Rule**
State change	$a \mapsto b$ $\widetilde{x} \cdot a \cdot \widetilde{y} \mapsto \widetilde{x} \cdot b \cdot \widetilde{y}$
Complexation	$a \,|\, b \mapsto c$ $\widetilde{x} \cdot a \cdot \widetilde{y} \,|\, b \mapsto \widetilde{x} \cdot c \cdot \widetilde{y}$
Decomplexation	$c \mapsto a \,|\, b$ $\widetilde{x} \cdot c \cdot \widetilde{y} \mapsto \widetilde{x} \cdot a \cdot \widetilde{y} \,|\, b$
Catalysis	$c \,|\, P_1 \mapsto c \,|\, P_2$ (where $P_1 \mapsto P_2$ is the catalysed event)
Membrane crossing	$a \,|\, (\widetilde{x})^L \rfloor X \mapsto (\widetilde{x})^L \rfloor (a \,|\, X)$ $(\widetilde{x})^L \rfloor (a \,|\, X) \mapsto a \,|\, (\widetilde{x})^L \rfloor X$ $\widetilde{x} \cdot a \cdot \widetilde{y} \,|\, (\widetilde{z})^L \rfloor X \mapsto (\widetilde{z})^L \rfloor (\widetilde{x} \cdot a \cdot \widetilde{y} \,|\, X)$ $(\widetilde{z})^L \rfloor (\widetilde{x} \cdot a \cdot \widetilde{y} \,|\, X) \mapsto \widetilde{x} \cdot a \cdot \widetilde{y} \,|\, (\widetilde{z})^L \rfloor X$
Catalyzed membrane crossing	$a \,|\, (b \cdot \widetilde{x})^L \rfloor X \mapsto (b \cdot \widetilde{x})^L \rfloor (a \,|\, X)$ $(b \cdot \widetilde{x})^L \rfloor (a \,|\, X) \mapsto a \,|\, (b \cdot \widetilde{x})^L \rfloor X$ $\widetilde{x} \cdot a \cdot \widetilde{y} \,|\, (b \cdot \widetilde{z})^L \rfloor X \mapsto (b \cdot \widetilde{z})^L \rfloor (\widetilde{x} \cdot a \cdot \widetilde{y} \,|\, X)$ $(b \cdot \widetilde{z})^L \rfloor (\widetilde{x} \cdot a \cdot \widetilde{y} \,|\, X) \mapsto \widetilde{x} \cdot a \cdot \widetilde{y} \,|\, (b \cdot \widetilde{z})^L \rfloor X$
Membrane joining	$(\widetilde{x})^L \rfloor (a \,|\, X) \mapsto (a \cdot \widetilde{x})^L \rfloor X$ $(\widetilde{x})^L \rfloor (\widetilde{y} \cdot a \cdot \widetilde{z} \,|\, X) \mapsto (\widetilde{y} \cdot a \cdot \widetilde{z} \cdot \widetilde{x})^L \rfloor X$
Catalyzed	$(b \cdot \widetilde{x})^L \rfloor (a \,|\, X) \mapsto (a \cdot b \cdot \widetilde{x})^L \rfloor X$

Table 1. Guidelines for the abstraction of biomolecular events into CLS.

geographical frontiers (expressing the possible environments for a migrant population). In the application of Section 5, we focus on a level in which a looping sequence might either denote the surface of a cell (when modelling the red blood cells) or of an organic tissue containing these kind of cells.

We give now some examples of biomolecular events of interest and their description in CLS. The simplest kind of event is the change of state of an elementary object. Then, we consider interactions between molecules: in particular complexation, decomplexation and catalysis. These interactions may involve single elements of non-elementary molecules (DNA and RNA strands, and proteins). Moreover, there can be interactions between membranes and molecules: in particular a molecule may cross or join a membrane.

Table 1 lists some guidelines (taken from [8]) for the abstraction into CLS rules of biomolecular events. Entities are associated with CLS terms: elementary objects are modelled as alphabet symbols, non-elementary objects as CLS sequences and membranes as looping sequences. Biomolecular events are associated with CLS rewrite rules.

3 A Type Discipline for Required and Excluded Elements

We classify elements in $\mathcal{E}$ with *basic types*. 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. We assume a fixed typing Γ for the elements in $\mathcal{E}$.

For each basic type $\mathtt{t}$ we assume to have a pair of sets of basic 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 basic type $\mathtt{t}$ requires the presence of elements whose basic type is in $\mathtt{R_t}$ and forbids the presence of elements whose basic type is in $\mathtt{E_t}$. We consider only *local* properties: elements influence each other if they are either present in the same compartment or one is present in the looping sequence and the other is present in the inner compartment of a containment operator.

The type system derives the set of types of patterns (and therefore also terms), checking that the constraints imposed by the required and excluded sets are satisfied. Types are pairs $(\mathtt{P}, \mathtt{R})$: $\mathtt{P}$ is the set of basic types of elements that are *present* in the top-level compartment of the term, and $\mathtt{R}$ is the set of basic types of elements that are *required* to fulfill the requirements of the elements present in the top-level compartment of the term. The set of *excluded* elements for a given set $\mathtt{P}$ of present elements is given by $\mathtt{E_P} = \bigcup_{\mathtt{t} \in \mathtt{P}} \mathtt{E_t}$.

Types, $(\mathtt{P}, \mathtt{R})$, are well formed if the required types, $\mathtt{R}$, are required by the present elements, $\mathtt{P}$, and the constraints on required and excluded elements are not contradictory. Pairs of types are compatible if required and excluded types are compatible with the union of their present types. Pairs of compatible types can be combined.

Definition 3.1 (Auxiliary definitions).

- A type $(\mathtt{P}, \mathtt{R})$ is *well formed* if
  - $\mathtt{R} \subseteq \bigcup_{\mathtt{t} \in \mathtt{P}} \mathtt{R_t}$, and
  - $\mathtt{P} \cap \mathtt{E_P} = \mathtt{P} \cap \mathtt{R} = \mathtt{R} \cap \mathtt{E_P} = \emptyset$.
- 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
  - $\mathtt{E_P} \cap \mathtt{P}' = \mathtt{E_P} \cap \mathtt{R}' = \emptyset$, and
  - $\mathtt{E_{P'}} \cap \mathtt{P} = \mathtt{E_{P'}} \cap \mathtt{R} = \emptyset$.
- 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

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

Given a basic type $\mathtt{t}$, $(\{\mathtt{t}\}, \mathtt{R_t})$, is well formed, indeed, it is the type of a compartment containing only elements of type $\mathtt{t}$. A requirement we could have asked is that of repellency being symmetric, that is: for all $\mathtt{t}, \mathtt{t}'$, if $\mathtt{t} \in \mathtt{E_{t'}}$, then $\mathtt{t}' \in \mathtt{E_t}$. However, such a requirement would not change the type system, since compatibility of types encompasses this property.

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 3. It is easy to verify that, if we start from

$$\Delta, \rho : (\mathtt{P}, \mathtt{R}) \vdash \rho : (\mathtt{P}, \mathtt{R}) \quad (\mathsf{Tvar}) \qquad \Delta \vdash \epsilon : (\emptyset, \emptyset) \quad (\mathsf{Teps}) \qquad \frac{a : \mathtt{t} \in \Gamma}{\Delta \vdash a : (\{\mathtt{t}\}, \mathtt{R}_\mathtt{t})} \, (\mathsf{Tel})$$

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

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

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

Fig. 3. Typing Rules

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 basis.

All the rules are trivial except for the last one which types containment operators. In this rule we can put a pattern P inside a containment operator with looping sequence SP only if all the types required from P are provided by SP. This is because the elements present in the inner compartment can not interact with the elements present outside the looping sequence.

It is easy to verify that the type system of Figure 3 enjoys weakening, i.e. that $\Delta \vdash P : (\mathtt{P}, \mathtt{R})$ and $\Delta \subseteq \Delta'$ imply $\Delta' \vdash P : (\mathtt{P}, \mathtt{R})$.

Moreover also the following substitution properties will be handy.

Lemma 3.2. If $\Delta \vdash C[P] : (\mathtt{P}, \mathtt{R})$ then

1. $\Delta \vdash P : (\mathtt{P}', \mathtt{R}')$ for some $(\mathtt{P}', \mathtt{R}')$, and
2. $\Delta, X : (\mathtt{P}', \mathtt{R}') \vdash C[X] : (\mathtt{P}, \mathtt{R})$, and
3. if P' is such that $\Delta \vdash P' : (\mathtt{P}', \mathtt{R}')$, then $\Delta \vdash C[P'] : (\mathtt{P}, \mathtt{R})$.

Proof. Easy by induction on the definition of contexts. □

We are interested in applying reduction rules only to correct terms, whose type is well formed and whose requirements are completely satisfied. More formally:

Definition 3.3. A term T is *correct* if $\vdash T : (\mathtt{P}, \emptyset)$ for some $\mathtt{P}$.

Example 3.4. Assuming $\mathcal{E} = \{a, b, c, d\}$ $\Gamma = \{a : \mathtt{t}_a, b : \mathtt{t}_b, c : \mathtt{t}_c, d : \mathtt{t}_d\}$ $\mathtt{R}_b = \{\mathtt{t}_c\}$ $\mathtt{R}_c = \{\mathtt{t}_a\}$ $\mathtt{E}_d = \{\mathtt{t}_b, \mathtt{t}_c\}$ $\mathtt{R}_a = \mathtt{R}_d = \mathtt{E}_a = \mathtt{E}_b = \mathtt{E}_c = \emptyset$ and using the rules in Figure 3, the terms in lines $(*)$ and $(**)$ of Example 2.8 have type $(\{\mathtt{t}_a, \mathtt{t}_d\}, \emptyset)$, so they are correct terms. However, the term in line $(***)$, does not have a type, since the element b is in the same compartment of the element d, but the basic type of b is in the set of the elements excluded by the presence of the basic type of d.

Rules such that the left-hand-side and the right-hand-side patterns have the same type do not change the type of terms to which they are applied.

Definition 3.5 (Δ-safe rules). A rewrite rule $P_1 \mapsto P_2$ is Δ-*safe* if $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$ and $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$ for some $(\mathtt{P}, \mathtt{R})$.

When we apply a Δ-safe rule to a term we need to choose an instantiation which agrees with Δ, i.e. σ must replace the variables in the domain of Δ as prescribed by Δ. More formally:

Definition 3.6. An instantiation σ *agrees* with a basis Δ (notation $\sigma \in \Sigma_\Delta$) if $\rho : (\mathtt{P}, \mathtt{R}) \in \Delta$ implies $\vdash \sigma(\rho) : (\mathtt{P}, \mathtt{R})$.

As expected agreement between substitutions and basis assures type preservation, as proved in the following lemma.

Lemma 3.7. If $\sigma \in \Sigma_\Delta$, then $\vdash P\sigma : (\mathtt{P}, \mathtt{R})$ if and only if $\Delta \vdash P : (\mathtt{P}, \mathtt{R})$.

Proof. ($\Leftarrow$) By induction on $\Delta \vdash P : (\mathtt{P}, \mathtt{R})$. Consider the last applied rule.

- If the rule is (Tvar), the proof follows from $\sigma \in \Sigma_\Delta$. For rules (Teps), (Tel) the fact that P is a term implies that $P\sigma = P$ and, moreover, it is typable from the empty environment.
- Rule (Tseq). In this case $P = SP{\cdot}SP'$, $(\mathtt{P}, \mathtt{R}) = (\mathtt{P}'', \mathtt{R}'') \sqcup (\mathtt{P}', \mathtt{R}')$, $\Delta \vdash SP : (\mathtt{P}'', \mathtt{R}'')$, $\Delta \vdash SP' : (\mathtt{P}', \mathtt{R}')$ and $(\mathtt{P}'', \mathtt{R}'') \bowtie (\mathtt{P}', \mathtt{R}')$. By induction hypotheses, $\vdash SP\,\sigma : (\mathtt{P}'', \mathtt{R}'')$ and $\vdash SP'\,\sigma : (\mathtt{P}', \mathtt{R}')$. Therefore, since $SP\,\sigma{\cdot}SP'\,\sigma = (SP{\cdot}SP')\,\sigma$, applying rule (Tseq) we conclude $\vdash (SP{\cdot}SP')\,\sigma : (\mathtt{P}, \mathtt{R})$.
- For rules (Tpar), (Tcomp) the proof is similar.

($\Rightarrow$) By induction on P.

- If $P = \rho$, the proof follows from $\sigma \in \Sigma_\Delta$. If $P = \epsilon$, or $P = a$, by weakening.
- Let P be $SP{\cdot}SP'$. Since $(SP{\cdot}SP')\,\sigma = SP\,\sigma{\cdot}SP'\,\sigma$, the fact that $\vdash (SP{\cdot}SP')\,\sigma : (\mathtt{P}, \mathtt{R})$ implies that the last applied rule must be (Tseq). Therefore, $(\mathtt{P}, \mathtt{R}) = (\mathtt{P}'', \mathtt{R}'') \sqcup (\mathtt{P}', \mathtt{R}')$, $(\mathtt{P}'', \mathtt{R}'') \bowtie (\mathtt{P}', \mathtt{R}')$, $\vdash SP\,\sigma : (\mathtt{P}'', \mathtt{R}'')$, and $\vdash SP'\,\sigma : (\mathtt{P}', \mathtt{R}')$. By induction hypothesis on SP and SP' we get $\Delta \vdash SP : (\mathtt{P}'', \mathtt{R}'')$, and $\Delta \vdash SP' : (\mathtt{P}', \mathtt{R}')$. Applying rule (Tseq) we conclude $\Delta \vdash SP{\cdot}SP' : (\mathtt{P}, \mathtt{R}) \sqcup (\mathtt{P}', \mathtt{R}')$.
- If $P = P' \mid P''$ or $P = (SP)^L \rfloor P'$ the proof is similar.

□

Since Δ-safe rules do not modify the type of a term, typing creation and degradation of elements is not possible. Moreover, also movements of elements between membranes are very limited.

Example 3.8. Assuming the sets in Example 3.4 and the basis

$$\Delta = \{\widetilde{x} : (\{\mathtt{t}_b, \mathtt{t}_c\}, \emptyset), Y : (\emptyset, \emptyset)\}$$

the rule in Example 2.5 is a Δ-safe rule, because the left and the right side of the rule have the same type:

$$\Delta \vdash (a \cdot \widetilde{x})^L \rfloor (b \mid Y) : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset) \quad \Delta \vdash b \mid (a \cdot \widetilde{x})^L \rfloor Y : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$$

However, using the basis

$$\Delta' = \{\widetilde{x} : (\emptyset, \emptyset), Y : (\{\mathtt{t}_a\}, \emptyset)\}$$

the same rule is not a Δ'-safe rule, because left-hand-side and right-hand-side of the rule do not have the same type:

$$\Delta' \vdash (a \cdot \widetilde{x})^{L} \rfloor (b \,|\, Y) : (\{\mathtt{t}_a\}, \emptyset) \quad \Delta' \vdash b \,|\, (a \cdot \widetilde{x})^{L} \rfloor Y : (\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$$

To permit the application of rules that may introduce/remove/move elements preserving safety, we introduce a restriction on rules based on the context of application rather than, as for Δ-safety, the type of patterns involved in the rule. To this extent we first characterize contexts by the types of terms that may fill their hole, and then rules by the types of terms that their application produces.

Definition 3.9 (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 $\mathtt{P}'$.

The above notion guarantees that filling a context with a term whose type is OK for the context we obtain a correct term: 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 3.10 (Δ-$(\mathtt{P}, \mathtt{R})$-safe rules). A rewrite rule $P_1 \mapsto P_2$ is Δ-$(\mathtt{P}, \mathtt{R})$-*safe* if $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$.

To ensure correctness, we can apply a rule to a typed term only if the instance of the pattern on the right-hand-side of the rule has a type that is OK for the context. This choice makes possible typing creation and degradation of elements. On the other hand, at every application of the rule we must check if the type of the right-hand-side of the rule is OK for the context.

Example 3.11. Assuming the sets in Example 3.4 and the basis

$$\Delta = \{\widetilde{x} : (\emptyset, \emptyset), Y : (\emptyset, \emptyset)\}$$

the rule in Example 2.5 is Δ-$(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$-safe, because the right side of the rule has type $(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$.
Let C_1 be the context $(a \cdot c) \,|\, \Box$, the type $(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$ is OK for C_1, since

$$X : (\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\}) \vdash C_1[X] : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$$

and so we can apply the rule in Example 2.5.

Instead, the type $(\{\mathtt{t}_a, \mathtt{t}_b\}, \emptyset)$ is not OK for the context $C_2 = a \,|\, \Box$, since

$$X : (\{\mathtt{t}_a, \mathtt{t}_b\}, \emptyset) \vdash C_2[X] : (\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$$

and so we cannot apply the rule in Example 2.5.

Since both Δ-safe and Δ-$(\mathtt{P}, \mathtt{R})$-safe rules preserve correctness, our semantics uses both.

Definition 3.12 (Typed Semantics). Given a finite set of rewrite rules $\mathcal{R}$, the *typed semantics* of CLS is the least relation closed with respect to $\equiv$ satisfying the following (sets of) rules:

$$\frac{\Re = P_1 \mapsto P_2 \in \mathcal{R} \text{ is a } \Delta\text{-safe rule} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma_\Delta \quad C \in \mathcal{C}}{C[P_1\sigma] \Longrightarrow C[P_2\sigma]} \,(\Re\text{-}\Delta)$$

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

	Δ	type of $\mathtt{P}_1\sigma$	type of $\mathtt{P}_2\sigma$	type of $C[\mathtt{P}_2\sigma]$
(1)	$\widetilde{x} : (\{\mathtt{t}_b, \mathtt{t}_c\}, \{\mathtt{t}_a\})\ Y : (\emptyset, \emptyset)$	$(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$	$(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$	$(\{\mathtt{t}_a, \mathtt{t}_d\}, \emptyset)$
(2)	$\widetilde{x} : (\{\mathtt{t}_c\}, \{\mathtt{t}_a\})\ Y : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$	$(\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset)$	$(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$	$(\{\mathtt{t}_a, \mathtt{t}_d\}, \emptyset)$
(3)	$\widetilde{x} : (\emptyset, \emptyset)\ Y : (\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset)$	$(\{\mathtt{t}_a\}, \emptyset)$	$(\{\mathtt{t}_a, \mathtt{t}_b\}, \emptyset)$	—

Fig. 4. Basis and Typings of Example 3.14

Reduction preserves correctness, making possible typing of creation and degradation.

Theorem 3.13. If $\vdash T : (\mathtt{P}, \emptyset)$ and $T \Longrightarrow T'$, then $\vdash T' : (\mathtt{P}', \emptyset)$ for some $\mathtt{P}'$.

Proof. We analyse the two sets of rules of the semantics separately. Let $\Re = P_1 \mapsto P_2$.

Rules ($\Re$-Δ) From Definition 3.12, $T = C[P_1\sigma]$ and $T' = C[P_2\sigma]$ and $\sigma \in \Sigma_\Delta$. By hypothesis $\vdash C[P_1\sigma] : (\mathtt{P}, \emptyset)$. Therefore, Lemma 3.2.(1) implies $\vdash P_1\sigma : (\mathtt{P}', \mathtt{R}')$ for some $(\mathtt{P}', \mathtt{R}')$, and from Lemma 3.7 we derive $\Delta \vdash P_1 : (\mathtt{P}', \mathtt{R}')$. By Definition 3.5, we get that $\Delta \vdash P_2 : (\mathtt{P}', \mathtt{R}')$. Applying Lemma 3.7, we derive that $\vdash P_2\sigma : (\mathtt{P}', \mathtt{R}')$. Finally, from Lemma 3.2.(3) we obtain $\vdash C[P_2\sigma] : (\mathtt{P}, \emptyset)$.

Rules ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$) From Definition 3.10, we have that $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$. Lemma 3.7 and $\sigma \in \Sigma_\Delta$ imply that $\vdash P_2\sigma : (\mathtt{P}, \mathtt{R})$. Since, from Definition 3.12, $(\mathtt{P}, \mathtt{R})$ is OK for C, we get that $X : (\mathtt{P}, \mathtt{R}) \vdash C[X] : (\mathtt{P}', \emptyset)$ for some $\mathtt{P}'$. Therefore, by Lemma 3.2.(3) we conclude that $\vdash C[P_2\sigma] : (\mathtt{P}', \emptyset)$.

□

Example 3.14. Using the sets in Example 3.4 we can study the behaviour of the term in Example 2.8. That is the evolution of the initial term due to the application of the ($\Re$-Δ) and ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$) rules where $\Re = P_1 \mapsto P_2$, with

- $P_1 = (a \cdot \widetilde{x})^L \rfloor (b \,|\, Y)$, and
- $P_2 = b \,|\, (a \cdot \widetilde{x})^L \rfloor Y$

and $\Delta_{(1)}$ is Δ of the first line in Fig. 4, etc. Rule $P_1 \mapsto P_2$ is a $\Delta_{(1)}$-safe rule, since $\Delta_{(1)} \vdash P_1 : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$, and $\Delta_{(1)} \vdash P_2 : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$. Therefore, applying rule ($\Re$-Δ) we get

$$d \,|\, (a)^L \rfloor ((a \cdot c)^L \rfloor ((a \cdot b \cdot c)^L \rfloor (b))) \Longrightarrow d \,|\, (a)^L \rfloor ((a \cdot c)^L \rfloor (b \,|\, (a \cdot b \cdot c)^L \rfloor \epsilon))$$

the reduction in line ($*$) of Example 3.4.

The rule $P_1 \mapsto P_2$ is not a $\Delta_{(2)}$-safe rule, since $\Delta_{(2)} \vdash P_1 : (\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset)$, and $\Delta_{(2)} \vdash P_2 : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$. However, $P_1 \mapsto P_2$ is a $\Delta_{(2)}$-$(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$-safe rule and the context of the reduction $C_{(2)}$, in the second line of Fig. 2, is OK for $(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$. So, applying rule ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$) we get

$$d \,|\, (a)^L \rfloor ((a \cdot c)^L \rfloor (b \,|\, (a \cdot b \cdot c)^L \rfloor \epsilon)) \Longrightarrow d \,|\, (a)^L \rfloor (b \,|\, (a \cdot c)^L \rfloor ((a \cdot b \cdot c)^L \rfloor \epsilon))$$

the reduction in line ($**$) of Example 3.4.

Finally, for the third reduction neither of the conditions holds.

- Firstly, $P_1 \mapsto P_2$ is not $\Delta_{(3)}$-safe, since $\Delta_{(3)} \vdash P_1 : (\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset)$ and $\Delta_{(3)} \vdash P_2 : (\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$.
- Secondly, even though $P_1 \mapsto P_2$ is $\Delta_{(3)}$-$(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$-safe, the context $C_{(3)} = d\,|\,\Box$ is not OK for $(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$.

Indeed the term in line $(***)$ of Example 3.4 cannot be typed.

It is possible to prove that given $\Re = P_1 \mapsto P_2$ if the requirements for applying rule ($\Re$-Δ) are satisfied, then also the requirements for applying rule ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$) are satisfied.

Theorem 3.15. If $P_1 \mapsto P_2$ is a Δ-safe rule and $P_1\sigma \not\equiv \epsilon$ and $\sigma \in \Sigma_\Delta$ and $C \in \mathcal{C}$ and $\vdash C[P_1\sigma] : (\mathtt{P}', \emptyset)$, then there is a type $(\mathtt{P}, \mathtt{R})$ OK for C such that $P_1 \mapsto P_2$ is a Δ-$(\mathtt{P}, \mathtt{R})$-safe rule.

Proof. From the hypothesis that $P_1 \mapsto P_2$ is a Δ-safe rule, and Definition 3.5 we have that $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$, and $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$. Therefore, from Definition 3.10, $P_1 \mapsto P_2$ is a Δ-$(\mathtt{P}, \mathtt{R})$-safe rule. From $\sigma \in \Sigma_\Delta$, $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$ and Lemma 3.7 we derive that $\vdash P_1\sigma : (\mathtt{P}, \mathtt{R})$. The hypothesis $\vdash C[P_1\sigma] : (\mathtt{P}', \emptyset)$ and Lemma 3.2.(2) imply that $X : (\mathtt{P}, \mathtt{R}) \vdash C[X] : (\mathtt{P}', \emptyset)$, and so $(\mathtt{P}, \mathtt{R})$ is OK for C. □

Given a set of rewrite rules $\mathcal{R}$, the previous theorem proves that if a term is reducible with the typed semantics whose reductions use only ($\Re$-Δ) rules ($\Re \in \mathcal{R}$), then the term is also reducible with the typed semantics whose reductions use only ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$) rules ($\Re \in \mathcal{R}$). Example 3.14 shows that the vice versa is not true. Moreover, a term reducible with the typed semantics whose reductions use both sets of rules ($\Re$-Δ) and ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$) is also reducible with the typed semantics whose reductions use only rules ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$). The advantage to have both sets of rules is that, to check that $\Re = P_1 \mapsto P_2$ may be applied to a well-typed term using rules ($\Re$-Δ) is more efficient that checking that rules ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$) are applicable. This is because for both kinds of rules, once we have the substitution σ derived from the matching of the pattern P_1 with the term, we have to show that $\Delta \vdash P_1 : (\mathtt{P}_1, \mathtt{R}_1)$, and $\Delta \vdash P_2 : (\mathtt{P}_2, \mathtt{R}_2)$ where $\sigma \in \Sigma_\Delta$. Moreover, for rules ($\Re$-Δ) we have to see whether $P_2\,\sigma$ has the same type as $P_1\,\sigma$, whereas for rules ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$), in addition to find the type of $P_2\,\sigma$, we have to see whether this type is OK for the context. This implies to derive the type of the context C. In the following section, we will show how to use type inference to provide an algorithm for the typed semantics of Definition 3.12, that takes advantage from these considerations.

4 Type Inference

The definition of typed semantics (Definition 3.12) is not effective, since we do not know how to choose Δ for ($\Re$-Δ) rules and $\Delta, \mathtt{P}, \mathtt{R}$ for ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$) rules. In the present section we define inference rules for principal typing [21] in order to derive which rules are Δ-safe and which ones are Δ-$(\mathtt{P}, \mathtt{R})$-safe, where the choices of $\Delta, \mathtt{P}, \mathtt{R}$ are guided by the term we want to reduce. This will allow us to get an algorithm for checking the applicability of reduction rules to typed terms preserving typing.

We convene that for each variable $x \in \mathcal{X}$ there is an *e-type variable* φ_x ranging over basic types, and for each variable $\eta \in \mathcal{TV} \cup \mathcal{SV}$ there are two variables ϕ_η, ψ_η

$$\vdash \epsilon : \emptyset; (\emptyset, \emptyset); \emptyset \quad (\mathsf{Reps}) \qquad \frac{a : \mathtt{t} \in \Gamma}{\vdash a : \emptyset; (\{\mathtt{t}\}, \mathtt{R}_{\mathtt{t}}); \emptyset}\,(\mathsf{Rel})$$

$$\vdash x : \{x : (\{\varphi_x\}, \mathtt{R}_{\varphi_x})\}; (\{\varphi_x\}, \mathtt{R}_{\varphi_x}); \emptyset \quad (\mathsf{Rvar}_1)$$
$$\vdash \eta : \{\eta : (\phi_\eta, \psi_\eta)\}; (\phi_\eta, \psi_\eta); \emptyset \quad (\mathsf{Rvar}_2)$$

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

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

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

Fig. 5. Inference Rules for Principal Typing

(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. We denote by μ a generic p-type, r-type or e-type variable.
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:

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

The rules for inferring principal typings 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 5 gives these inference rules.

Example 4.1. We can use the inference rules in Figure 5 to infer the types of the patterns of the rule in Example 2.5, where, again, we assume the basic types of Example 3.4, obtaining

$$\vdash P_1 : \Theta; (\{\mathtt{t}_a\} \cup \phi_{\widetilde{x}}, \psi_{\widetilde{x}} \setminus (\{\mathtt{t}_b\} \cup \phi_Y)); \Xi_1$$
$$\vdash P_2 : \Theta; (\{\mathtt{t}_a, \mathtt{t}_b\} \cup \phi_{\widetilde{x}}, \{\mathtt{t}_c\} \cup (\psi_{\widetilde{x}} \setminus \phi_Y)); \Xi_2$$

where

$$\begin{aligned}
\Theta \quad &= \quad \{\ \widetilde{x} : (\phi_{\widetilde{x}}, \psi_{\widetilde{x}}), \quad X : (\phi_Y, \psi_Y)\ \} \\
\Xi_1 \quad &= \quad \{\ (\{\mathtt{t}_a\}, \emptyset) \bowtie (\phi_{\widetilde{x}}, \psi_{\widetilde{x}}), \quad (\{\mathtt{t}_b\}, \{\mathtt{t}_c\}) \bowtie (\phi_Y, \psi_Y), \\
&\qquad (\{\mathtt{t}_a\} \cup \phi_{\widetilde{x}}, \psi_{\widetilde{x}}) \bowtie (\{\mathtt{t}_b\} \cup \phi_Y, \{\mathtt{t}_c\} \cup \psi_Y), \quad \{\mathtt{t}_c\} \cup \psi_Y \subseteq \{\mathtt{t}_a\} \cup \phi_{\widetilde{x}}\ \} \\
\Xi_2 \quad &= \quad \{\ (\{\mathtt{t}_a\}, \emptyset) \bowtie (\phi_{\widetilde{x}}, \psi_{\widetilde{x}}), \quad (\{\mathtt{t}_a\} \cup \phi_{\widetilde{x}}, \psi_{\widetilde{x}}) \bowtie (\phi_Y, \psi_Y), \quad \psi_Y \subseteq \{\mathtt{t}_a\} \cup \phi_{\widetilde{x}}, \\
&\qquad (\{\mathtt{t}_b\}, \{\mathtt{t}_c\}) \bowtie (\{\mathtt{t}_a\} \cup \phi_{\widetilde{x}}, \psi_{\widetilde{x}} \setminus \phi_Y)\ \}
\end{aligned}$$

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

Proof. By induction on derivations, and by cases on the last applied rule.

- For rules (Reps), (Rel), (Rvar_1), and (Rvar_2) the result is trivial.
- Rule (Rseq). In this case the conclusion of the rule is $\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Phi, \Psi) \sqcup (\Phi', \Psi'); \Xi \cup \Xi' \cup \{(\Phi, \Psi) \bowtie (\Phi', \Psi')\}$, and the assumptions are $\vdash SP : \Theta; (\Phi, \Psi); \Xi$ and $\vdash SP' : \Theta'; (\Phi', \Psi'); \Xi'$. Since m satisfies Ξ and Ξ', by induction hypothesis, and weakening, we derive that $\mathsf{m}(\Theta \cup \Theta') \vdash SP : (\mathsf{m}(\Phi), \mathsf{m}(\Psi))$ and $\mathsf{m}(\Theta \cup \Theta') \vdash SP' : (\mathsf{m}(\Phi'), \mathsf{m}(\Psi'))$. Moreover, since m satisfies $\{(\Phi, \Psi) \bowtie (\Phi', \Psi')\}$, we have that $(\mathsf{m}(\Phi), \mathsf{m}(\Psi)) \bowtie (\mathsf{m}(\Phi'), \mathsf{m}(\Psi'))$. So rule (Tseq) can be applied, and $\mathsf{m}(\Theta \cup \Theta') \vdash SP{\cdot}SP' : (\mathsf{m}(\mathtt{P}), \mathsf{m}(\mathtt{R})) \sqcup (\mathsf{m}(\mathtt{P}'), \mathsf{m}(\mathtt{R}'))$.
- For rules (Rpar), and (Rcomp) the result can be proved like for rule (Rseq).

□

Theorem 4.3 (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 $\Delta \supseteq \mathsf{m}(\Theta)$, $\mathtt{P} = \mathsf{m}(\Phi)$, $\mathtt{R} = \mathsf{m}(\Psi)$.

Proof. By induction on the derivation of $\Delta \vdash P : (\mathtt{P}, \mathtt{R})$.

- If the last rule of the derivation is (Teps), (Tel), or (Tvar) the result is obvious. Note that, for (Tvar) in the inference we distinguish the case of element variables (from sequence or term variables).
- Rule (Tseq). The conclusion of the rule is $\Delta \vdash SP{\cdot}SP' : (\mathtt{P}, \mathtt{R}) \sqcup (\mathtt{P}', \mathtt{R}')$, and the assumptions are $\Delta \vdash SP : (\mathtt{P}, \mathtt{R})$, $\Delta \vdash SP' : (\mathtt{P}', \mathtt{R}')$ and the condition $(\mathtt{P}, \mathtt{R}) \bowtie (\mathtt{P}', \mathtt{R}')$. By induction hypothesis, there are Θ, Φ, Ψ, Ξ, Θ', Φ', Ψ', Ξ' such that $\vdash SP : \Theta; (\Phi, \Psi); \Xi$ and $\vdash SP' : \Theta'; (\Phi', \Psi'); \Xi'$. These are the assumptions of rule (Rseq), whose conclusion is $\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Phi, \Psi) \sqcup (\Phi', \Psi'); \Xi \cup \Xi' \cup \{(\Phi, \Psi) \bowtie (\Phi', \Psi')\}$. Moreover, by induction there is a type mapping m' satisfying Ξ such that $\Delta \supseteq \mathsf{m}'(\Theta)$, $\mathtt{P} = \mathsf{m}'(\Phi)$ and $\mathtt{R} = \mathsf{m}'(\Psi)$, and there is a type mapping m'' satisfying Ξ' such that $\Delta \supseteq \mathsf{m}''(\Theta')$, $\mathtt{P}' = \mathsf{m}''(\Phi')$ and $\mathtt{R}' = \mathsf{m}''(\Psi')$. Therefore, we derive $\Delta \supseteq \mathsf{m}'(\Theta) \cup \mathsf{m}''(\Theta')$ and $(\mathtt{P}, \mathtt{R}) \sqcup (\mathtt{P}', \mathtt{R}') = (\mathsf{m}'(\Phi), \mathsf{m}'(\Psi)) \sqcup (\mathsf{m}''(\Phi'), \mathsf{m}''(\Psi'))$. Since the basis $\mathsf{m}'(\Theta)$ and $\mathsf{m}''(\Theta')$ are both subsets of the same basis Δ, then for all the (e-type, p-type or r-type) variables μ such that $\mu \in dom(\mathsf{m}') \cap dom(\mathsf{m}'')$ we get $\mathsf{m}'(\mu) = \mathsf{m}''(\mu)$. Therefore the mapping m
$$\mathsf{m}(\mu) = \begin{cases} \mathsf{m}'(\mu) & \text{if } \mu \in dom(\mathsf{m}') \\ \mathsf{m}''(\mu) & \text{if } \mu \in dom(\mathsf{m}'') \end{cases}$$
is well defined.
Moreover, since m satisfies Ξ, Ξ' and $(\Phi, \Psi) \bowtie (\Phi', \Psi')$, then m satisfies also all the constraints of the conclusion of the rule (Rseq).
- If the last rule is (Tpar) or (Tcomp) the proof is similar.

□

Now, we put our inference rules at work in order to decide the applicability of typed reduction rules, for both Δ-safe and Δ-$(\mathtt{P},\mathtt{R})$-safe rules.

To decide applicability of Δ-safe rules, we characterize Δ-safe rules.

Lemma 4.4 (Characterization of Δ-safe rules). A rule $P_1 \mapsto P_2$ is a Δ-safe 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 $\Xi_1 \cup \Xi_2 \cup \{\Phi_1 = \Phi_2\} \cup \{\Psi_1 = \Psi_2\}$, where $\vdash P_1 : \Theta_1; (\Phi_1, \Psi_1); \Xi_1$ and $\vdash P_2 : \Theta_2; (\Phi_2, \Psi_2); \Xi_2$.

Proof. ($\Leftarrow$) Since $\vdash P_1 : \Theta_1; (\Phi_1, \Psi_1); \Xi_1$, $\vdash P_2 : \Theta_2; (\Phi_2, \Psi_2); \Xi_2$ and m satisfies Ξ_1 and Ξ_2, applying Theorem 4.2 we derive $\mathsf{m}(\Theta_1) \vdash P_1 : (m(\Phi_1), m(\Psi_1))$, and $\mathsf{m}(\Theta_2) \vdash P_2 : (m(\Phi_2), m(\Psi_2))$. From the definition of m, we have that $\mathsf{m}(\Theta_2) \subseteq \Delta$ and $\mathsf{m}(\Theta_2) \subseteq \Delta$, and by weakening we derive that $\Delta \vdash P_1 : (m(\Phi_1), m(\Psi_1))$ and $\Delta \vdash P_2 : (m(\Phi_2), m(\Psi_2))$. Moreover, from the fact that m satisfies $\{\Phi_1 = \Phi_2\} \cup \{\Psi_1 = \Psi_2\}$, we have that $\mathsf{m}(\Phi_1) = \mathsf{m}(\Phi_2) = \mathtt{P}$ and $\mathsf{m}(\Psi_1) = \mathsf{m}(\Psi_2) = \mathtt{R}$. Therefore, $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$ and $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$, and $P_1 \mapsto P_2$ is a Δ-safe rule.

($\Rightarrow$) Since $P_1 \mapsto P_2$ is a Δ-safe rule, we have that $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$ and $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$. From Theorem 4.3, applied to $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$, we derive that $\vdash P_1 : \Theta_1; (\Phi_1, \Psi_1); \Xi_1$ and there is a type mapping m' satisfying Ξ_1 such that $\Delta \supseteq \mathsf{m}'(\Theta_1)$, $\mathtt{P} = \mathsf{m}'(\Phi_1)$, $\mathtt{R} = \mathsf{m}'(\Psi_1)$. Applying Theorem 4.3 to $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$ we derive that $\vdash P_2 : \Theta_2; (\Phi_2, \Psi_2); \Xi_2$ and there is a type mapping m'' satisfying Ξ_2 such that $\Delta \supseteq \mathsf{m}''(\Theta_2)$, $\mathtt{P} = \mathsf{m}''(\Phi_2)$, $\mathtt{R} = \mathsf{m}''(\Psi_2)$. Since the basis $\mathsf{m}'(\Theta_1)$ and $\mathsf{m}''(\Theta_2)$ are both subsets of Δ, then, the mapping m defined by

$$\mathsf{m}(\mu) = \begin{cases} \mathsf{m}'(\mu) & \text{if } \mu \in dom(\mathsf{m}') \\ \mathsf{m}''(\mu) & \text{if } \mu \in dom(\mathsf{m}'') \end{cases}$$

is well defined. Moreover, m satisfies $\Xi_1 \cup \Xi_2$, and since $\mathsf{m}'(\Phi_1) = \mathtt{P} = \mathsf{m}''(\Phi_2)$, $\mathsf{m}'(\Psi_1) = \mathtt{R} = \mathsf{m}''(\Psi_2)$, then m also satisfies $\{\Phi_1 = \Phi_2\} \cup \{\Psi_1 = \Psi_2\}$.

□

Example 4.5. Using Lemma 4.4, we can see that the constraints making Δ-safe the rule in Example 2.5 are

$$\{\mathtt{t}_a\} \cup \phi_{\widetilde{x}} = \{\mathtt{t}_a, \mathtt{t}_b\} \cup \phi_{\widetilde{x}} \quad \psi_{\widetilde{x}} \setminus (\{\mathtt{t}_b\} \cup \phi_{\widetilde{x}}) = \{\mathtt{t}_c\} \cup (\psi_{\widetilde{x}} \setminus \phi_Y)$$

and the constraints in the sets Ξ_1 and Ξ_2 of Example 4.1.

To decide applicability of Δ-$(\mathtt{P},\mathtt{R})$-safe rules, we characterize the OK relation and Δ-$(\mathtt{P},\mathtt{R})$-safe rules.

Regarding 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* the subterm of the context including the hole and the part of the context affecting the type of the hole. The following definition formalises this notion.

Definition 4.6. The *core of the context* C (notation $\mathsf{core}(C)$) is defined by:

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

Remark that core is unambiguously defined, since every context can be split in an unique way into one of the three shapes of the previous definition.

Lemma 4.7 (Characterization of 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

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

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

Proof. ($\Leftarrow$) Lemma 3.2.(1) and $\vdash C[T] : (\mathtt{P}_0, \emptyset)$ imply that all subterms of $\mathsf{core}(C)[X]$ are typable, i.e. that there are $\mathtt{P}_1, \mathtt{R}_1, \mathtt{P}'_1, \mathtt{R}'_1, \mathtt{P}_2, \mathtt{R}_2, \mathtt{P}'_2, \mathtt{R}'_2$ such that $\vdash T_1 : (\mathtt{P}_1, \mathtt{R}_1)$, $\vdash S_1 : (\mathtt{P}'_1, \mathtt{R}'_1)$, $\vdash T_2 : (\mathtt{P}_2, \mathtt{R}_2)$, $\vdash S_2 : (\mathtt{P}'_2, \mathtt{R}'_2)$ in the last case of the definition of $\mathsf{core}(C)[X]$, and suitable subsets of these typing judgements in the other two cases.
By Definition 4.6 we have the following cases.

- $C = \mathsf{core}(C)$ and
  - either $\mathsf{core}(C) = \Box \,|\, T_1$ and $\Phi = \phi_X \cup \mathtt{P}_1$ and $\Psi = \psi_X \cup \mathtt{R}_1$,
  - or $\mathsf{core}(C) = (S_1)^L \rfloor (\Box \,|\, T_1) \,|\, T_2$ and $\Phi = \mathtt{P}'_1 \cup \mathtt{P}_2$ and $\Psi = (\mathtt{R}'_1 \cup \mathtt{R}_2) \setminus (\phi_X \cup \mathtt{P}_1)$.

  Since m satisfies $\{\Psi = \emptyset \text{ if } \phi_X \text{ or } \psi_X \text{ occurs in } \Psi\}$, then $\mathsf{m}(\Psi) = \emptyset$. From Theorem 4.2, since $\vdash \mathsf{core}(C)[X] : \{X : (\phi_X, \psi_X)\}; (\Phi, \Psi); \Xi$ and m satisfies Ξ, we derive that $X : (\mathtt{P}, \mathtt{R}) \vdash \mathsf{core}(C)[X] : (\mathsf{m}(\Phi), \emptyset)$. Moreover, since $C[X] = \mathsf{core}(C)[X]$, we have that $X : (\mathtt{P}, \mathtt{R}) \vdash C[X] : (\mathsf{m}(\Phi), \emptyset)$. Therefore, the type $(\mathtt{P}, \mathtt{R})$ is OK for the context C.
- $\mathsf{core}(C) = (S_2)^L \rfloor ((S_1)^L \rfloor (\Box \,|\, T_1) \,|\, T_2)$ and $\Phi = \mathtt{P}'_2$ and $\Psi = \mathtt{R}'_2 \setminus (\mathtt{P}'_1 \cup \mathtt{P}_2)$.
  From $\vdash C[T] : (\mathtt{P}_0, \emptyset)$ by Lemma 3.2.(1) and .(2) we get $X : (\mathtt{P}', \mathtt{R}') \vdash \mathsf{core}(C)[X] : (\mathtt{P}'', \mathtt{R}'')$ for some $\mathtt{P}', \mathtt{R}', \mathtt{P}'', \mathtt{R}''$. This implies by the Completeness Theorem (Theorem 4.3) that there is a mapping m' such that $\mathsf{m}'(\mathtt{P}'_2) = \mathtt{P}''$ and $\mathsf{m}'(\mathtt{R}'_2 \setminus (\mathtt{P}'_1 \cup \mathtt{P}_2)) = \mathtt{R}''$. Since $\mathtt{P}'_2$ and $\mathtt{R}'_2 \setminus (\mathtt{P}'_1 \cup \mathtt{P}_2)$ do not contain variables, we get $\mathtt{P}'_2 = \mathtt{P}''$ and $\mathtt{R}'_2 \setminus (\mathtt{P}'_1 \cup \mathtt{P}_2) = \mathtt{R}''$, independently from the types assumed for the variable X. This implies by Lemma 3.2.(3) and .(2) $X : (\mathtt{P}, \mathtt{R}) \vdash C[X] : (\mathtt{P}_0, \emptyset)$, so we conclude that $(\mathtt{P}, \mathtt{R})$ is OK for the context C.

($\Rightarrow$) By Definition 3.9, since $(\mathtt{P}, \mathtt{R})$ is OK for C, then $X : (\mathtt{P}, \mathtt{R}) \vdash C[X] : (\mathtt{P}', \emptyset)$ for some $\mathtt{P}'$. Theorem 4.3 implies that $\vdash C[X] : \Theta'; (\Phi', \Psi'); \Xi'$ and there is a type mapping m that satisfies Ξ' and such that $\{X : (\mathtt{P}, \mathtt{R})\} \supseteq \mathsf{m}(\Theta')$, $\mathsf{m}(\Phi') = \mathtt{P}$, $\mathsf{m}(\Psi') = \emptyset$. By definition $\Theta' = \{X : (\phi_X, \psi_X)\}$, so we get $\mathsf{m}(\phi_X) = \mathtt{P}$ and

$\mathsf{m}(\psi_X) = \mathtt{R}$. Being $\mathsf{core}(C)[X]$ a subterm of $C[X]$ by Lemma 3.2.(1) we get $X : (\mathtt{P}, \mathtt{R}) \vdash \mathsf{core}(C)[X] : (\mathtt{P}'', \mathtt{R}')$ for some $\mathtt{P}'', \mathtt{R}'$. Theorem 4.3 implies that $\vdash \mathsf{core}(C)[X] : \{X : (\phi_X, \psi_X)\}; (\Phi, \Psi); \Xi$ and by construction $\Xi \subseteq \Xi'$, so m satisfies also Ξ. If $\mathsf{core}(C) = C$, then $\Psi = \Psi'$, which implies $\mathsf{m}(\Psi) = \emptyset$. Otherwise neither ϕ_X nor ψ_X occurs in Ψ.

□

It is easy to check that if $\mathsf{core}(C) \equiv (S_2)^L \rfloor ((S_1)^L \rfloor (\Box \,|\, T_1) \,|\, T_2)$, 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 to prove that C is OK we have to verify the following six constraints:

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

The set of constraints is smaller when the core context has one of the simpler shapes.

Example 4.8. Using Lemma 4.7, the constraints making the type associated with the p-type and r-type variable X OK for the contexts in Example 2.8 are:

$$\begin{array}{ll} (A) & (\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset) \bowtie (\phi_X, \psi_X) \quad \psi_X \subseteq \{\mathtt{t}_a, \mathtt{t}_c\} \\ (B) & (\{\mathtt{t}_a\}, \emptyset) \bowtie (\phi_X, \psi_X) \quad \psi_X \subseteq \{\mathtt{t}_a\} \\ (C) & (\{\mathtt{t}_d\}, \emptyset) \bowtie (\phi_X, \psi_X) \quad \psi_X = \emptyset. \end{array}$$

Lemma 4.9 (Characterization of Δ-$(\mathtt{P}, \mathtt{R})$-safe rules). A rule $P_1 \mapsto P_2$ is Δ-$(\mathtt{P}, \mathtt{R})$-safe 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 $\Xi_2 \cup \{\Phi_2 = \mathtt{P}, \Psi_2 = \mathtt{R}\}$, where $\vdash P_2 : \Theta_2; (\Phi_2, \Psi_2); \Xi_2$.

Proof. ($\Leftarrow$) Let $\vdash P_2 : \Theta_2; (\Phi_2, \Psi_2); \Xi_2$ and m satisfies $\Xi_2 \cup \{\Phi_2 = \mathtt{P}, \Psi_2 = \mathtt{R}\}$. From Theorem 4.2 we derive that $\mathsf{m}(\Theta) \vdash P_2 : (\mathtt{P}, \mathtt{R})$. By definition of m we get $\mathsf{m}(\Theta_2) = \Delta$. Therefore $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$, and $P_1 \mapsto P_2$ is a Δ-$(\mathtt{P}, \mathtt{R})$-safe rule.

($\Rightarrow$) Let $P_1 \mapsto P_2$ be a Δ-$(\mathtt{P}, \mathtt{R})$-safe rule, then $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$. From Theorem 4.3, we have that $\vdash P_2 : \Theta_2; (\Phi_2, \Psi_2); \Xi_2$, for some $\Theta_2, \Phi_2, \Psi_2, \Xi_2$, and there is a type mapping m' satisfying Ξ such that $\Delta \supseteq \mathsf{m}'(\Theta_2)$, $\mathtt{P} = \mathsf{m}'(\Phi_2)$, and $\mathtt{R} = \mathsf{m}'(\Psi_2)$. Therefore m' satisfies $\Xi_2 \cup \{\Phi_2 = \mathtt{P}, \Psi_2 = \mathtt{R}\}$. From definition of m, we get $\mathsf{m}(\Theta_2) = \Delta$, and since $\Delta \supseteq \mathsf{m}'(\Theta_2)$, also m satisfies $\Xi_2 \cup \{\Phi_2 = \mathtt{P}, \Psi_2 = \mathtt{R}\}$.

□

The previous lemmas imply the following theorem asserting the condition of applicability of the rewrite rules.

Theorem 4.10 (Applicability of rewrite rules). Let

$$\vdash P_1 : \Theta_1; (\Phi_1, \Psi_1); \Xi_1 \quad \text{and} \quad \vdash P_2 : \Theta_2; (\Phi_2, \Psi_2); \Xi_2 \quad \text{and}$$
$$\vdash \mathsf{core}(C)[X] : \{X : (\phi_X, \psi_X)\}; (\Phi_C, \Psi_C); \Xi_C \quad \text{and} \quad P_1\sigma \not\equiv \epsilon.$$

Then the rule $P_1 \mapsto P_2$ can be applied to the term $C[P_1\sigma]$ such that $\vdash C[P_1\sigma] : (\mathtt{P}_0, \emptyset)$ (for some $\mathtt{P}_0$) 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

(*a*) either the set of constraints $\Xi_1 \cup \Xi_2 \cup \{\Phi_1 = \Phi_2\} \cup \{\Psi_1 = \Psi_2\}$,
(*b*) or the set of constraints $\Xi_2 \cup \Xi_C \cup \{\Phi_2 = \phi_X, \Psi_2 = \psi_X\} \cup \{\Psi_C = \emptyset \mid \text{ if } \phi_X \text{ or } \psi_X \text{ occurs in } \Psi_C\}$.

Proof. We define the basis Δ as follows:

$x : (\{\mathtt{t}\}, \mathtt{R}_\mathtt{t}) \in \Delta$ if $\sigma(x) : \mathtt{t} \in \Gamma$, and
$\eta : (\mathtt{P}', \mathtt{R}') \in \Delta$ if $\vdash \sigma(\eta) : (\mathtt{P}', \mathtt{R}')$.

In this way we get that $\sigma \in \Sigma_\Delta$ and the type mapping m is such that:

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

Let $\Re = P_1 \mapsto P_2$.

($\Leftarrow$) If the mapping m satisfies the the set of constraints $\Xi_1 \cup \Xi_2 \cup \{\Phi_1 = \Phi_2\} \cup \{\Psi_1 = \Psi_2\}$, then by Lemma 4.4 the rule $P_1 \mapsto P_2$ is Δ-safe and we get $C[P_1\sigma] \Longrightarrow C[P_2\sigma]$ by applying rule ($\Re$-Δ).
If the mapping m satisfies the the set of constraints $\Xi_2 \cup \Xi_C \cup \{\Phi_2 = \phi_X, \Psi_2 = \psi_X\} \cup \{\Psi_C = \emptyset \mid \text{ if } \phi_X \text{ or } \psi_X \text{ occurs in } \Psi_C\}$, then by Lemma 4.7 the context C is OK for $(\mathtt{P}, \mathtt{R})$ and by Lemma 4.9 the rule $P_1 \mapsto P_2$ is Δ-$(\mathtt{P}, \mathtt{R})$-safe; we get $C[P_1\sigma] \Longrightarrow C[P_2\sigma]$ by applying rule ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$).

($\Rightarrow$) If $C[P_1\sigma] \Longrightarrow C[P_2\sigma]$ by applying rule ($\Re$-Δ), then the rule $P_1 \mapsto P_2$ is Δ-safe and then the mapping m satisfies the the set of constraints $\Xi_1 \cup \Xi_2 \cup \{\Phi_1 = \Phi_2\} \cup \{\Psi_1 = \Psi_2\}$ by Lemma 4.4.
If $C[P_1\sigma] \Longrightarrow C[P_2\sigma]$ by applying rule ($\Re$-Δ-$(\mathtt{P}, \mathtt{R})$), then the rule $P_1 \mapsto P_2$ is Δ-$(\mathtt{P}, \mathtt{R})$-safe and the context C is Ok for $(\mathtt{P}, \mathtt{R})$, then the mapping m satisfies the the set of constraints $\Xi_2 \cup \Xi_C \cup \{\Phi_2 = \phi_X, \Psi_2 = \psi_X\} \cup \{\Psi_C = \emptyset \mid \text{ if } \phi_X \text{ or } \psi_X \text{ occurs in } \Psi_C\}$ by Lemmas 4.7 and 4.9.

□

The mapping m may be easily defined from the derivation of a type for $P_1\sigma$, and the checking that m satisfies a set of constraints requires only some substitutions.

Note that the sets of constraints for typing the left-hand-side and the right-hand-side of Δ-safe rules, and the right-hand-side of Δ-$(\mathtt{P}, \mathtt{R})$-safe rules, can be inferred once

	$\phi_{\widetilde{x}}$	$\psi_{\widetilde{x}}$	ϕ_Y	ψ_Y	ϕ_X	ψ_X
(1)	$\mathtt{t}_b, \mathtt{t}_c$	$\mathtt{t}_a$	$\emptyset$	$\emptyset$	$\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c$	$\emptyset$
(2)	$\mathtt{t}_c$	$\mathtt{t}_a$	$\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c$	$\emptyset$	$\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c$	$\emptyset$
(3)	$\emptyset$	$\emptyset$	$\mathtt{t}_a$	$\emptyset$	$\mathtt{t}_a, \mathtt{t}_b$	$\mathtt{t}_c$

Fig. 6. Type mappings of Example 4.11

for all. Instead, the set of constraints for typing the core context of an application of a Δ-$(\mathtt{P},\mathtt{R})$-safe rule has to be inferred when trying to apply the rule. However, as previously remarked, this set of constraints includes at most six constraints.

Theorem 3.15 implies that, during inference, we can first check if the rule is Δ-safe, using the constraints associated with the rule, and if this is not the case check whether the rule is a Δ-$(\mathtt{P},\mathtt{R})$-safe rule (for some $(\mathtt{P},\mathtt{R})$), and if the type is OK for the context. We can summarise the idea in the following algorithm, where Δ is defined in the proof of Theorem 4.10:

- In the initial phase, for every rule $\Re = P_1 \mapsto P_2 \in \mathcal{R}$, we infer $\vdash P_1 : \Theta; (\Phi_1, \Psi_1); \Xi_1$ and $\vdash P_2 : \Theta; (\Phi_2, \Psi_2); \Xi_2$.
- When trying to reduce the well-typed term $C[P_1\,\sigma]$ with $\Re = P_1 \mapsto P_2$, we first check whether the conditions of ($\Re$-Δ) hold, that is:

  1. we check whether the type mapping m, defined by

     $\mathsf{m}(\varphi_x) = \mathtt{t}$ if $\sigma(x) : \mathtt{t} \in \Gamma$,

     $\mathsf{m}(\phi_\eta) = \mathtt{P}'$ if $\vdash \sigma(\eta) : (\mathtt{P}', \mathtt{R}')$,

     $\mathsf{m}(\psi_\eta) = \mathtt{R}'$ if $\vdash \sigma(\eta) : (\mathtt{P}', \mathtt{R}')$,

     satisfies Ξ_2: if not $P_2\,\sigma$ is not well typed, and neither ($\Re$-Δ) nor ($\Re$-Δ-$(\mathtt{P},\mathtt{R})$) would be applicable ($C[P_1\,\sigma]$ is not reducible via $P_1 \mapsto P_2$);
  2. we check if m satisfies $\{\Phi_1 = \Phi_2\} \cup \{\Psi_1 = \Psi_2\}$. If this is the case, the rule is a Δ-safe rule, and then we can apply ($\Re$-Δ),
  3. otherwise we check whether the conditions of ($\Re$-Δ-$(\mathtt{P},\mathtt{R})$) hold, where $\mathtt{P} = \mathsf{m}(\Phi_2)$ and $\mathtt{R} = \mathsf{m}(\Psi_2)$; in order to do this

     (a) we infer $\vdash \mathsf{core}(C)[X] : \{X : (\phi_X, \psi_X)\}; (\Phi_C, \Psi_C); \Xi_C$, and

     (b) we check whether m satisfies

     $$\Xi_C \cup \{\Phi_2 = \phi_X, \Psi_2 = \psi_X\} \cup \{\Psi_C = \emptyset \text{ if } \phi_X \text{ or } \psi_X \text{ occurs in } \Psi_C\}.$$

     If this is the case, the context C is OK for $(\mathtt{P},\mathtt{R})$, so we can use rule ($\Re$-Δ-$(\mathtt{P},\mathtt{R})$), otherwise neither rule ($\Re$-Δ) nor ($\Re$-Δ-$(\mathtt{P},\mathtt{R})$) is applicable ($C[P_1\,\sigma]$ is not reducible via $P_1 \mapsto P_2$).

As we can see, the only check that the algorithm would not perform if we were to use only rules ($\Re$-Δ-$(\mathtt{P},\mathtt{R})$), instead of both sets of rules, is the one in point 2. But the fact that the rule is Δ-safe implies that we do not have to perform the checks that follows (points 3.a and 3.b). In particular inferring the type for the context would be not needed.

Example 4.11. We use the algorithm described above on the terms of Example 3.14: the constraints for Δ-safe rules and OK relations for the contexts are reported in Examples 4.5 and 4.8, respectively. The type mappings derived from the instantiation are reported in Fig. 6.

(1) The type mapping in line (1) of Fig. 6 satisfies
   - the constraints in Ξ_2 associated with P_2 (see Example 4.1) and
   - the constraints that make the rule a Δ-safe rule (see Example 4.5).

(2) The type mapping in line (2) of Fig. 6
   - satisfies the constraints in Ξ_2 associated with P_2,
   - does not satisfy the constraint $\{\mathtt{t}_a\} \cup \phi_{\widetilde{x}} = \{\mathtt{t}_a, \mathtt{t}_b\} \cup \phi_{\widetilde{x}}$ that make the rule a Δ-safe rule (see Example 4.5) because
   $$\{\mathtt{t}_a, \mathtt{t}_c\} \neq \{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}$$
   - satisfies the set of constraints (B) for the context (see Example 4.8), since
   $$(\mathtt{t}_a, \emptyset) \bowtie (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset) \quad \emptyset \subseteq \{\mathtt{t}_a\}$$

(3) The type mapping in line (3) of Fig. 6
   - satisfies the constraints in Ξ_2 associated with P_2,
   - does not satisfy the constraint $\{\mathtt{t}_a\} \cup \phi_{\widetilde{x}} = \{\mathtt{t}_a, \mathtt{t}_b\} \cup \phi_{\widetilde{x}}$ that make the rule a Δ-safe rule (see Example 4.5) because
   $$\{\mathtt{t}_a\} \neq \{\mathtt{t}_b, \mathtt{t}_c\}$$
   - does not satisfy the set of constraints (C) for the context (see Example 4.8), since
   $$(\{\mathtt{t}_d\}, \emptyset) \bowtie (\{\mathtt{t}_a, \mathtt{t}_b\}, \emptyset) \quad \text{does not hold.}$$

5 Example

A blood type is a classification of blood based on the presence or absence of inherited antigenic substances on the surface of red blood cells: these antigens are the A antigen and the B antigen. Blood type A contains only A antigens, blood type B contains only B antigens, blood type AB contains both and the blood type O contains none of them: this classification is called ABO blood type system.

The immune system will produce antibodies that can specifically bind to a blood group antigen that is not recognized as self: individuals of blood type A have Anti-B antibodies, individuals of blood type B have Anti-A antibodies, individuals of blood type O have both Anti-A and Anti-B antibodies, and individuals of blood type AB have none of them. These antibodies can bind to the antigens on the surface of the transfused red blood cells, often leading to the destruction of the cell: for this reason, it is vital that compatible blood is selected for transfusions.

Another antigen that refines the classification of blood types is the RhD antigen: if this antigen is present, the blood type is called positive, else it is called negative. Unlike the ABO blood classification, the RhD antigen is immunogenic, meaning that a person who is RhD negative is very likely to produce Anti-RhD antibodies when

Recipient	Donor							
	O-	O+	A-	A+	B-	B+	AB-	AB+
O-	√							
O+	√	√						
A-	√		√					
A+	√	√	√	√				
B-	√				√			
B+	√	√			√	√		
AB-	√		√		√		√	
AB+	√	√	√	√	√	√	√	√

Table 2. Red blood cell compatibility.

element	basic type	R set	E set
c	$\mathtt{t}_c$	$\emptyset$	$\emptyset$
a	$\mathtt{t}_a$	$\mathtt{t}_c$	$\emptyset$
b	$\mathtt{t}_b$	$\mathtt{t}_c$	$\emptyset$
r	$\mathtt{t}_r$	$\mathtt{t}_c$	$\emptyset$
$\bar{a}$	$\mathtt{t}_{\bar{a}}$	$\mathtt{t}_c$	$\mathtt{t}_a$
$\bar{b}$	$\mathtt{t}_{\bar{b}}$	$\mathtt{t}_c$	$\mathtt{t}_b$
$\bar{r}$	$\mathtt{t}_{\bar{r}}$	$\mathtt{t}_c$	$\mathtt{t}_r$
t	$\mathtt{t}_t$	$\emptyset$	$\emptyset$

Table 3. Elements, basic types, R and E sets for red blood cell compatibility.

exposed to the RhD antigen, but it is also common for RhD-negative individuals not to have Anti-RhD antibodies. All these aspects led to the red blood cell compatibility table in Table 2.

We want to study the possibility of blood transfusion in a system consisting of a set of closed tissues. These tissues, containing blood cells and antibodies according to the rules described above, can join each other and exemplify a transfusion of different blood types. We model a red blood cell as a looping sequence containing the element c on the surface and, depending on the blood type, the elements a, b and r as the A antigen, the B antigen and the RhD antigen, respectively. We represent the antibodies as the single elements $\bar{a}$, $\bar{b}$ and $\bar{r}$, modelling, respectively, the Anti-A, Anti-B and Anti-RhD antibodies. Finally, we model a tissue (which can contains the red cells) as a looping sequence having only the element t on the surface. To avoid undesirable behaviours, using CLS without types, we must write as many rules as the different combinations of the different blood types shown in Table 2. Using the typed extension of CLS, according to the antigen and antibodies requirements and exclusions, we just create the basic types shown in Table 3 and we can use the single rule

$$(t)^L \rfloor X \mid (t)^L \rfloor Y \mapsto (t)^L \rfloor (X \mid Y)$$

to model tissues transfusion.

Let the system be a set of eight tissues, containing each possible recipient combination of blood with antibodies:

$$(t)^L \rfloor ((c)^L \rfloor \epsilon \mid \bar{a} \mid \bar{b} \mid \bar{r}) \mid$$
$$(t)^L \rfloor ((c \cdot a)^L \rfloor \epsilon \mid \bar{b} \mid \bar{r}) \mid (t)^L \rfloor ((c \cdot b)^L \rfloor \epsilon \mid \bar{a} \mid \bar{r}) \mid (t)^L \rfloor ((c \cdot r)^L \rfloor \epsilon \mid \bar{a} \mid \bar{b}) \mid$$
$$(t)^L \rfloor ((c \cdot a \cdot b)^L \rfloor \epsilon \mid \bar{r}) \mid (t)^L \rfloor ((c \cdot a \cdot r)^L \rfloor \epsilon \mid \bar{b}) \mid (t)^L \rfloor ((c \cdot b \cdot r)^L \rfloor \epsilon \mid \bar{a} \mid \bar{r}) \mid$$
$$(t)^L \rfloor ((c \cdot a \cdot b \cdot r)^L \rfloor \epsilon).$$

They cannot react with each other, because the antibodies of the ones exclude the antigens of the others. If in the system arrives a donor, as a tissue without antibodies, having blood type O-:

$$(t)^L \rfloor ((c)^L \rfloor \epsilon),$$

it can singularly react with each tissue, because it does not have antigens, whereas if in the system arrives a donor having blood type O+:

$$(t)^L \rfloor ((c \cdot r)^L \rfloor \epsilon),$$

it can singularly react with the tissues that do not contain the Anti-RhD antibody $\bar{r}$. As further example, if in the system arrives a donor having blood type A+:

$$(t)^L \rfloor ((c \cdot a \cdot r)^L \rfloor \epsilon),$$

it can singularly react with each tissue that does not contain Anti-RhD and Anti-A antibodies $\bar{r}$ and $\bar{a}$, so tissues containing A+ and AB+ blood types.

6 Conclusions

The most common approach of biologists to describe biological systems is based on the use of deterministic mathematical means (like, e.g., ODE), and makes it possible to abstractly reason on the behaviour of biological systems and to perform a quantitative *in silico* investigation. This kind of modelling, however, becomes more and more difficult, both in the specification phase and in the analysis processes, when the complexity of the biological systems taken into consideration increases. This has probably been one of the main motivations for the application of Computer Science formalisms to the description of biological systems [19].

In this paper we introduced a type system for CLS and used it to define a typed semantics in which the applicability of rules is determined by type conditions on the applied rules and on the context of application. We defined a type inference system and an algorithm to perform reductions.

As seen in Section 5, the use of a typed semantic for CLS permits to transfer the complexity of biological properties from rules to types, and so to study the behaviour of the systems using only simple and general rules: we focused on disciplines deriving by the requirement/exclusion of certain elements, even if 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 [5]. In particular, in [12], we show a simple example on how types could 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 [17].

In nature, request and repellency could be seen as practical *suggestions*, that if not followed could drive to undesirable behaviour, such as, like in our blood trans-

fusion example, the death of the system. In practice, one could not generally forbid terms having requests or repellency collisions: even if it is not desirable, one could accidentally transfuse incompatible blood types. On the contrary, our typed semantics completely exclude these kinds of situations. According to this idea, we can modify our typed semantics, allowing transitions which lead to untypable terms, but signalling that some errors, or some undesired states, as been reached. For example, we could modify the transitions driven by Δ-safe rules behaviour and Δ-$(\mathtt{P},\mathtt{R})$-safe rules behaviour, with the following two rules that raise an error when some undesired reduction is performed:

$$\frac{P_1 \mapsto P_2 \in \mathcal{R} \text{ is not a } \Delta\text{-}(\mathtt{P},\mathtt{R})\text{-safe rule} \qquad P_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma_\Delta \qquad C \in \mathcal{C}}{C[P_1\sigma] \xrightarrow{typabilityError} C[P_2\sigma]}$$

$$\frac{P_1 \mapsto P_2 \in \mathcal{R} \text{ is a } \Delta\text{-}(\mathtt{P},\mathtt{R})\text{-safe rule} \qquad P_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma_\Delta \qquad C \in \mathcal{C} \qquad (\mathtt{P},\mathtt{R}) \text{ is not OK for } C}{C[P_1\sigma] \xrightarrow{contextError} C[P_2\sigma]}$$

and modifying the algorithm in Section 4, raising an error in point 3.b. In this way the modeller knows that some unwanted behaviour is happening in the system and readjust it to avoid the undesired situations.

As highlighted from the previous considerations, the notion that a given element repels another is not fixed and immutable, but could arise, in an evolutionary way, from the failure of some rules. That is, instead of fixing the sets of required and excluded elements once and for all, in our reductions we could have sets that get modified by the (failure or success) of rule applications. The new sets could be used later on to influence the reductions of the system.

References

[1] Biocham. available at `http://contraintes.inria.fr/BIOCHAM/`.

[2] R. Alur, C. Belta, V. Kumar, and M. Mintz. Hybrid modeling and simulation of biomolecular networks. In *Hybrid Systems: Computation and Control*, volume 2034 of *LNCS*, pages 19–32. Springer, 2001.

[3] B. Aman, M. Dezani-Ciancaglini, and A. Troina. Type Disciplines for Analysing Biologically Relevant Properties. In *MeCBIC'08*, volume 227 of *ENTCS*, pages 97–111. Elsevier, 2009.

[4] R. Barbuti, A. Maggiolo-schettini, and P. Milazzo. Extending the calculus of looping sequences to model protein interaction at. In *ISBRA'07*, volume 4463 of *LNBI*, pages 638–649. Springer, 2006.

[5] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, P. Tiberi, and A. Troina. Stochastic calculus of looping sequences for the modelling and simulation of cellular pathways. *Transactions on Computational Systems Biology*, IX:86–113, 2008.

[6] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. Bisimulation congruences in the calculus of looping sequences. In *ICTAC'06*, volume 4281 of *LNCS*, pages 93–107. Springer, 2006.

[7] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. A calculus of looping sequences for modelling microbiological systems. *Fundamenta Informaticæ*, 72(1–3):21–35, 2006.

[8] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. Bisimulations in

calculi modelling membranes. *Formal Aspects of Computing*, 20(4-5):351–377, 2008.

[9] S. Capecchi and A. Troina. Types for BioAmbients. In *FBTC'10*, volume 19 of *EPTCS*, pages 103–115, 2009.

[10] L. Cardelli, V. Danos, and V. Schachter Eds. Brane calculi - interactions of biological membranes. In *Computational Methods in Systems Biology*, volume 3082 of *LNCS*, pages 257–280. Springer, 2005.

[11] V. Danos and C. Laneve. Core formal molecular biology. In *ESOP'03*, volume 2618 of *LNCS*, pages 302–318, 2003.

[12] M. Dezani-Ciancaglini, P. Giannini, and A. Troina. A Type System for a Stochastic CLS. In *MeCBIC'09*, volume 11 of *EPTCS*, pages 91–106, 2009.

[13] M. Dezani-Ciancaglini, P. Giannini, and A. Troina. A Type System for Required/Excluded Elements in CLS. In *DCM'09*, volume 9 of *EPTCS*, pages 38–48, 2009.

[14] F. Fages and S. Soliman. Abstract interpretation and types for systems biology. *Theoretical Computer Science*, 403(1):52–70, 2008.

[15] H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid Petri Net representation of gene regulatory networks. In *PSB'00*, pages 338–349. World Scientific Press, 2000.

[16] P. Milazzo. *Qualitative and quantitative formal modeling of biological systems*. PhD thesis, University of Pisa, 2007.

[17] G. Păun. *Membrane Computing. An Introduction*. Springer-Verlag, 2002.

[18] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. Bioambients: An abstraction for biological compartments. *Theoretical Computer Science*, 325(1):141–167, 2004.

[19] A. Regev and E. Shapiro. Cells as computation. *Nature*, 419(6905):343, September 2002.

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

[21] J. Wells. The Essence of Principal Typings. In *ICALP'02*, volume 2380 of *LNCS*, pages 913–925. Springer-Verlag, 2002.