Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

ENUMERATED TYPE SEMANTICS FOR THE CALCULUS OF LOOPING SEQUENCES [*]

Livio Bioglio[1]

Abstract. The calculus of looping sequences is a formalism for describing the evolution of biological systems by means of term rewriting rules. In this paper we enrich this calculus with a type discipline which preserves some biological properties depending on the minimum and the maximum number of elements of some type requested by the present elements. The type system enforces these properties and typed reductions guarantee that evolution preserves them. As an example, we model the hemoglobin structure and the equilibrium between cell death and division: typed reductions prevent undesirable behaviors.

1991 Mathematics Subject Classification. 03B15, 68Q42, 68Q55, 68Q65, 92C42.

1. Introduction

In the last few years, formalisms for the description and analysis of biological systems have been introduced and investigated, see [14]. These formalisms allow the study of biological systems by using methods, such as static analysis, model checking or quantitative computer simulations, which are practically unknown to biologists. Among the many formalisms that have either been applied to or inspired from biological systems, the most notable are: automata-based models [1, 10], rewrite systems [8, 12], and process calculi [7, 13–15]. Automata-based models permit the direct use of many verification tools such as model checkers; rewrite systems describe biological systems with a notation that can be easily understood by biologists; by means of process calculi the behavior of systems can be studied componentwise.

In [4, 5, 11], Milazzo et al. introduced a formalism, called Calculus of Looping Sequences (CLS for short), for describing biological systems and their evolution. CLS combines rewrite systems, being based on term rewriting, and process calculi,

[*] *This research is founded by the BioBITs Project (*Converging Technologies *2007, area: Biotechnology-ICT), Regione Piemonte.*

[1] Dipartimento di Informatica, Università di Torino e-mail: `biogliol@di.unito.it`

© EDP Sciences 1999

using features such as a commutative parallel composition operator, and semantic notions such as bisimulations.

Homeostasis is the property of a system that regulates its internal environment and tends to maintain stable conditions that are optimal for survival: when this equilibrium is disturbed, built-in regulatory devices respond to establish again the balance. Different living organisms have some homeostatic mechanism to maintain some conditions in specific ranges: the human body, like in all the warm-blooded animals, maintain a near-constant body temperature using mechanisms such as vasodilation and vasoconstriction, and microorganisms maintain the iron presence above a minimum level to maintain life but up to a maximum level to avoid iron toxicity. In biology a huge number of elements are made up of a certain number of different subcomponents: proteins are composed by different domains, some proteins are multimers, rybosomes are a mixture of RNA and proteins, etc. Monomers are molecules that may become chemically bounded to other monomers to form a polymer: for example, antibodies can be monomers, dimers or pentamers, the Triose phosphate isomerase, an enzyme essential for efficient energy production, is a dimer of identical subunits, and the Glutamate dehydrogenase 1, a mitochondrial matrix enzyme with a key role in the nitrogen and glutamate metabolism, is a hexamer.

In [2,3,6,9] different type disciplines for CLS are presented. The type discipline in [6, 9] preserves some biological properties deriving from the fact that certain elements may require the presence and the absence of others. With the type system introduced in [6,9], it is not possible to model constraints involved in homeostasis and polymers, because not only elements require the presence or absence of others, but they require that the numbers of these elements are in given intervals. In the type system introduced in this paper, we assume that for each element we fix the minimum and the maximum number of other elements that the element requires. We enrich CLS with a type discipline and typed reductions that guarantee the soundness of reduction rules with respect to the properties of biological systems deriving from the minimum and the maximum requested numbers of elements. This type discipline is a generalization of the one in [6, 9].

The remainder of the paper is organized as follows. In Section 2 we introduce the calculus of looping sequences. In Section 3 we develop the type discipline for the minimum and the maximum requested 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 constraints of rewrite rules, and check their applicability. In Section 5 we use our typing discipline to describe the hemoglobin structure and to regulate the equilibrium between cell death and division. Finally, in Section 6 we draw some conclusions.

2. Overview on Calculus of Looping Sequences

In this paper we refer to the definition of CLS formalism in [4].
We assume a possibly infinite alphabet $\mathcal{E}$ of symbols ranged over by $a, b, c, \ldots$.

Definition 1 (Patterns)**.** Patterns P *and* sequence patterns SP *of* CLS are given by the following grammar:

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

where a is a generic element of $\mathcal{E}$, and X, $\widetilde{x}$ and x are term variables, sequence variables and element variables, respectively. Terms are patterns without variables.

In CLS we have a sequencing operator $\_ \cdot \_$, a looping operator $(\_)^L$, a parallel composition operator $\_ \,|\, \_$ and a containment operator $\_ \rfloor \_$. Sequencing can be used to concatenate elements of the alphabet $\mathcal{E}$. The empty sequence 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 pattern, and we call this binary operator *loop*. A *compartment* is any parallel composition of one or more terms. Given a loop $(S)^L \rfloor P$, the compartment P is called the *inner compartment* of the looping sequence. We denote by χ a generic variable, i.e. χ stands for X, $\widetilde{x}$ or x.
In CLS we may have syntactically different terms representing the same structure. We introduce a structural congruence relation to identify such terms.

Definition 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:*

$$\begin{array}{c} S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3 \qquad S \cdot \epsilon \equiv_S \epsilon \cdot S \equiv_S S \\ S_1 \equiv_S S_2 \;\; \textit{implies} \;\; S_1 \equiv_T S_2 \;\; \textit{and} \;\; (S_1)^L \rfloor P \equiv_T (S_2)^L \rfloor P \\ P_1 \,|\, P_2 \equiv_T P_2 \,|\, P_1 \qquad P_1 \,|\, (P_2 \,|\, P_3) \equiv_T (P_1 \,|\, P_2) \,|\, P_3 \qquad P \,|\, \epsilon \equiv_T P \\ (\epsilon)^L \rfloor \epsilon \equiv_T \epsilon \qquad (S_1 \cdot S_2)^L \rfloor P \equiv_T (S_2 \cdot S_1)^L \rfloor P \end{array}$$

In the following, for simplicity, we will use $\equiv$ in place of $\equiv_T$.

A containment operator makes the inner compartment invisible from the outside. 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* P if $P \equiv P' \,|\, P''$ for some P', P'' and either $P' = S$ or $P' = (S)^L \rfloor \_$ for some S, and in both cases a is present in S.

Starting from a term, its behavior depends on rewrite rules: they are pairs of patterns, 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. Using variables, a rule will be applicable to all terms which can be obtained by properly instantiating its variables. An *instantiation* is a partial function that maps variables to terms, preserving their kinds: term variables, sequence variables and element variables are mapped into terms, sequences and elements, respectively. Given a pattern P, let $P\sigma$ denote the term obtained by replacing all occurrences of each variable χ appearing in P by the corresponding term $\sigma(\chi)$. We denote by Σ the set of all the possible instantiations, and by $Var(P)$ the set of variables appearing in P. Now we can formally define rewrite rules.

Definition 3 (Rewrite Rules)**.** *A rewrite rule, $\Re$, is a pair of patterns, denoted by $P_1 \mapsto P_2$, where $P_1 \not\equiv \epsilon$ and such that $Var(P_2) \subseteq Var(P_1)$.*

Example 2.1. *Examples of rewrite rules are*

$$(1)\ a \mapsto b \quad \textit{and} \quad (2)\ (d)^L \rfloor (b \,|\, X) \mapsto b \,|\, (d)^L \rfloor X$$

Rule (1) *says that one element a turns into one element b, and rule* (2) *says that one element b exits from a looping sequence containing only the element d. A rule can also be a pair of terms, like rule* (1)*, because terms are patterns without variables.*

A rewrite rule $P_1 \mapsto P_2$ states that a term $P_1\sigma$, obtained by instantiating variables in P_1 by some instantiation function $\sigma \in \Sigma$, can be transformed into the term $P_2\sigma$.

Rewrite rules can be applied to terms only if they occur in a legal context:

Definition 4 (Contexts)**.** Contexts C *are defined as:*

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

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

By definition, every context contains a single hole $\square$. Given $C_1, C_2 \in \mathcal{C}$, $C_1[C_2]$ denotes the context obtained by replacing $\square$ with C_2 in C_1. Given $C \in \mathcal{C}$ and a term T, $C[T]$ denotes the term obtained by replacing $\square$ with T in C.

The semantics of CLS is defined as follows.

Definition 5 (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:*

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

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

Example 2.2. *Starting from the term*

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

we can apply the rules in Example 2.1, obtaining, among the others, the following reductions

$$\begin{array}{llllll}
(d)^L \rfloor (c\,|\,a\,|\,a\,|\,a\,|\,a) & \rightarrow & (d)^L \rfloor (c\,|\,a\,|\,a\,|\,a\,|\,b) & \rightarrow & (d)^L \rfloor (c\,|\,a\,|\,a\,|\,b\,|\,b) \\
 & \rightarrow & b\,|\; (d)^L \rfloor (c\,|\,a\,|\,a\,|\,b) & \rightarrow & b\,|\; (d)^L \rfloor (c\,|\,a\,|\,b\,|\,b) \\
 & \rightarrow & b\,|\,b\,|\; (d)^L \rfloor (c\,|\,a\,|\,b) & \rightarrow & b\,|\,b\,|\,b\,|\; (d)^L \rfloor (c\,|\,a) \\
 & \rightarrow & b\,|\,b\,|\,b\,|\; (d)^L \rfloor (c\,|\,b) & \rightarrow & b\,|\,b\,|\,b\,|\,b\,|\; (d)^L \rfloor (c)
\end{array}$$

3. Type Discipline

Let $\mathscr{T}$ be a finite set of *basic types*: we classify every element in $\mathcal{E}$ with a unique element of $\mathscr{T}$. We use Γ to denote this classification. When there is no ambiguity, we denote the type associated with an element a by $\mathtt{t}_a$. In general, different elements can have the same basic type, so for example $\mathtt{t}_a = \mathtt{t}_b$ can hold. We assume the existence of two functions, $mn : \mathscr{T} \times \mathscr{T} \rightarrow \mathbb{N}$ and $mx : \mathscr{T} \times \mathscr{T} \rightarrow \mathbb{N} + \{\infty\}$ for every ordered pair of basic types $(\mathtt{t}_1, \mathtt{t}_2)$. These functions indicate the minimum and maximum (also infinity) number of elements of basic type $\mathtt{t}_2$ that can be present with one or more elements of basic type $\mathtt{t}_1$. We use infinity to mean the absence of a maximum limit, not to mean that there are infinite elements. For example, $mn(\mathtt{t}_a, \mathtt{t}_b) = 3$ means that if some elements of basic type $\mathtt{t}_a$ are present, then there must be also present at least 3 elements of basic type $\mathtt{t}_b$, and $mx(\mathtt{t}_a, \mathtt{t}_b) = 5$ means that if some elements of basic type $\mathtt{t}_a$ are present, then there can be also present at most 5 elements of basic type $\mathtt{t}_b$. If we consider both constraints, then the number of elements of basic type $\mathtt{t}_b$ in presence of some elements of basic type $\mathtt{t}_a$ must be between 3 and 5. We also consider $mn(\mathtt{t}, \mathtt{t})$ and $mx(\mathtt{t}, \mathtt{t})$, i.e. the minimum and maximum number of elements of basic type $\mathtt{t}$ allowed by the presence of one element of basic type $\mathtt{t}$. We assume that both do not include the presence of the element of basic type $\mathtt{t}$: for example, $mn(\mathtt{t}, \mathtt{t}) = 1$ means that one element of type $\mathtt{t}$ requires at least another element of the same basic type, and $mx(\mathtt{t}, \mathtt{t}) = 3$ means that one element of type $\mathtt{t}$ tolerates at most 3 other elements of the same basic type.
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. Note that the elements in a looping sequence are influenced by the elements present in the same compartment and also by the elements present in its inner compartment. We will write 'an element a requires n elements b' if $n = mn(\mathtt{t}_a, \mathtt{t}_b)$, and 'an element a tolerates m elements b' if $m = mx(\mathtt{t}_a, \mathtt{t}_b)$. The functions mn and mx cannot be arbitrary, they must satisfy some consistency requirements.

Definition 6 (Consistent Basic Types)**.** *A system composed by a set of basic types $\mathscr{T}$ and the functions mn and mx is* consistent *if:*

(1) $\forall\ \mathtt{t}_1, \mathtt{t}_2 \in \mathscr{T} \quad mn(\mathtt{t}_1, \mathtt{t}_2) \leq mx(\mathtt{t}_1, \mathtt{t}_2)$;
(2) $\forall\ \mathtt{t}_1, \mathtt{t}_2 \in \mathscr{T} \quad mx(\mathtt{t}_1, \mathtt{t}_2) = 0 \implies mn(\mathtt{t}_2, \mathtt{t}_1) = 0$;
(3) $\forall\ \mathtt{t}_1, \mathtt{t}_2 \in \mathscr{T} \quad mn(\mathtt{t}_1, \mathtt{t}_2) > 0 \implies mx(\mathtt{t}_2, \mathtt{t}_1) > 0$.

The meaning of the conditions stated in Definition 6 is:

(1) the minimum number of elements of basic type $\mathtt{t}_2$ required by the elements of basic type $\mathtt{t}_1$ must be lower than the maximum number of elements of the same basic type $\mathtt{t}_2$ tolerated by the elements of basic type $\mathtt{t}_1$;
(2) if the elements of basic type $\mathtt{t}_1$ do not tolerate elements of basic type $\mathtt{t}_2$, then the elements of basic type $\mathtt{t}_2$ cannot require elements of basic type $\mathtt{t}_1$.
(3) if the elements of basic type $\mathtt{t}_1$ require the presence of a certain number of elements having basic type $\mathtt{t}_2$, then the elements of basic type $\mathtt{t}_2$ must tolerate elements of basic type $\mathtt{t}_1$.

Note that intolerance is not symmetric: the elements of a basic type $\mathtt{t}_1$ can tolerate elements of basic type $\mathtt{t}_2$ also if elements of basic type $\mathtt{t}_2$ are intolerant versus elements of basic type $\mathtt{t}_1$.

Example 3.1. *The system*

$$\mathscr{T} = \{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}$$

where mn, mx are:

mn	$\mathtt{t}_a$	$\mathtt{t}_b$	$\mathtt{t}_c$
$\mathtt{t}_a$	0	0	0
$\mathtt{t}_b$	0	0	1
$\mathtt{t}_c$	1	0	0

mx	$\mathtt{t}_a$	$\mathtt{t}_b$	$\mathtt{t}_c$
$\mathtt{t}_a$	∞	0	1
$\mathtt{t}_b$	∞	∞	∞
$\mathtt{t}_c$	∞	1	∞

is consistent, because every pair of basic types respects the constrains in Definition 6.

The present type discipline is a refinement of the $\mathtt{P}/\mathtt{R}$ type discipline for CLS proposed in [6,9]. In the $\mathtt{P}/\mathtt{R}$ type discipline each basic type $\mathtt{t}$ is associated with a pair of sets of basic types $(\mathtt{R}_\mathtt{t}, \mathtt{E}_\mathtt{t})$, where $\mathtt{t} \notin \mathtt{R}_\mathtt{t} \cup \mathtt{E}_\mathtt{t}$ and $\mathtt{R}_\mathtt{t} \cap \mathtt{E}_\mathtt{t} = \emptyset$, saying that the presence of elements of basic type $\mathtt{t}$ requires the presence of elements whose basic type belongs to $\mathtt{R}_\mathtt{t}$ and forbids the presence of elements whose basic type belongs to $\mathtt{E}_\mathtt{t}$. We can express the sets $\mathtt{R}$ and $\mathtt{E}$ of the $\mathtt{P}/\mathtt{R}$ type discipline by means of the functions mn and mx. Given a basic type $\mathtt{t}$, we have $mn(\mathtt{t}, \mathtt{t}) = 0$ and $mx(\mathtt{t}, \mathtt{t}) = \infty$, for every basic type $\mathtt{t}' \in \mathtt{R}_\mathtt{t}$ we have $mn(\mathtt{t}, \mathtt{t}') = 1$ and $mx(\mathtt{t}, \mathtt{t}') = \infty$, whereas for every basic type $\mathtt{t}'' \in \mathtt{E}_\mathtt{t}$ we have $mn(\mathtt{t}, \mathtt{t}'') = mx(\mathtt{t}, \mathtt{t}'') = 0$.

Types are triples $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ of multisets over the set $\mathscr{T}$ of basic types, where $\mathtt{P}$ (*present-ms*) is the multiset of basic types of *present* elements (the elements present at the top level compartment of a pattern, i.e. in the outermost compartment), $\mathtt{L}$ (*at-least-ms*) is the multiset of the basic types still required by the present elements, and $\mathtt{M}$ (*at-most-ms*) is the multiset of the basic types still tolerated by the present elements. Some basic definitions about multisets, taken from [16] and extended with infinity, are reported in Figure 1.

Given a basic type $\mathtt{t}$, we define its *down-ms* $\mathtt{D}_\mathtt{t}$ as the multiset $\langle \mathscr{T}, f_{\mathtt{D}_\mathtt{t}} \rangle$, where $f_{\mathtt{D}_\mathtt{t}}(\mathtt{t}') = mn(\mathtt{t}, \mathtt{t}')$ (for $\mathtt{t}' \in \mathscr{T}$), and its *up-ms* $\mathtt{U}_\mathtt{t}$ as the multiset $\langle \mathscr{T}, f_{\mathtt{U}_\mathtt{t}} \rangle$, where $f_{\mathtt{U}_\mathtt{t}}(\mathtt{t}') = mx(\mathtt{t}, \mathtt{t}')$ (for $\mathtt{t}' \in \mathscr{T}$).

A type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is *well formed* if:

- **Multiset**: a *multiset* over a set D is a pair $\langle D, f\rangle$, where $f : D \to \mathbb{N} \cup \{\infty\}$ is a function, called *multiplicity function*.
- **Empty multiset**: a multiset $A = \langle D, f_A\rangle$ is the empty multiset, denoted by $\emptyset$, if $\forall\ e \in D \quad f_A(e) = 0$.
- **Infinite multiset**: a multiset $A = \langle D, f_A\rangle$ is the infinite multiset, denoted by D_∞, if $\forall\ e \in D \quad f_A(e) = \infty$.
- **Sub-multiset**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then A is a *sub-multiset* of B, denoted $A \subseteq B$, if $\forall\ e \in D \quad f_A(e) \leq f_B(e)$.
- **Sum**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then their *sum*, denoted $A \uplus B$, is the multiset $C = \langle D, f_C\rangle$ such that $\forall\ e \in D$: $f_C(e) = f_A(e) + f_B(e)$.
- **Removal**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then the *removal* of multiset B from A, denoted $A \ominus B$, is the multiset $C = \langle D, f_C\rangle$ such that $\forall\ e \in D \quad f_C(e) = max(f_A(e) - f_B(e), 0)$.
- **Union**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then their *union*, denoted $A \cup B$, is the multiset $C = \langle D, f_C\rangle$ such that $\forall\ e \in D \quad f_C(e) = max(f_A(e), f_B(e))$.
- **Intersection**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then their *intersection*, denoted $A \cap B$, is the multiset $C = \langle D, f_C\rangle$ such that $\forall\ e \in D \quad f_C(e) = min(f_A(e), f_B(e))$.
- We convene that $\forall\ m \in \mathbb{N} \cup \{\infty\}$, $\forall\ n \in \mathbb{N}$
  - $m \leq \infty$
  - $m + \infty = \infty$
  - $\infty - n = \infty$
  - $n - \infty = 0$.

FIGURE 1. Multiset Basic Definitions

- the multiset `L` is a subset of the multiset `M`, i.e. the multiplicity of every basic type in `L` is less than or equal to the multiplicity of the same basic type in `M`,
- the multiset `L` is contained in the union of the *down-ms* of the types in `P`,
- the sum of the multisets `P` and `M` is contained in the intersection of the *up-ms* of each basic type in `P`, taking into account the elements themselves.

For example, if a basic type in `P` requires 3 elements of basic type $\mathtt{t}_1$, and a different basic type in `P` requires 5 elements of the same basic type $\mathtt{t}_1$, the multiplicity of $\mathtt{t}_1$ in `L` can be any number less than or equal to 5, because there can be elements of basic type $\mathtt{t}_1$ which are present, but it cannot be greater than 5. Therefore the multiplicity of $\mathtt{t}_1$ in `L` must be not greater than the maximum of the *down-ms* of the basic types in `P`. In multiset theory, the maximum between multisets is their union, so the *at-least-ms* must be a subset of the union of the *down-ms* of all the basic types in the *present-ms*. Since the *up-ms* contains the maximum number of basic types still tolerated by the elements in the *present-ms*, the sum between the *at-most-ms* and the *present-ms* must be a subset of the intersection of the *up-ms* of all the basic types in the *present-ms*. We sum to the *up-ms* of a basic type `t` the basic type itself because by definition the *up-ms* of a basic type $\mathtt{t}$ does not take into account the presence of $\mathtt{t}$. For example, let $\mathtt{P}$ contain 2 elements of basic type $\mathtt{t}_1$, that tolerates 5 elements of the basic type $\mathtt{t}_1$ itself, and a basic type in $\mathtt{P}$ tolerates 7 elements of basic type $\mathtt{t}_1$. The sum of the multiplicities of $\mathtt{t}_1$ in $\mathtt{P}$ and $\mathtt{M}$ must be then the minimum between 6 and 7, i.e. 6. Since the multiplicity of $\mathtt{t}_1$ in $\mathtt{P}$ is 2, then the multiplicity of $\mathtt{t}_1$ in $\mathtt{M}$ must be less than or equal to 4.

Definition 7 (Well-Formed Types)**.** *A type* $(\mathtt{P},\mathtt{L},\mathtt{M})$ *is* well formed *if* $\mathtt{L} \subseteq \mathtt{M}$, $\mathtt{L} \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}} \mathtt{D}_\mathtt{t}$ *and* $\mathtt{P} \uplus \mathtt{M} \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}} (\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$.

In the following we will consider only well-formed types.
Summarizing, the (well-formed) type of a pattern is $(\mathtt{P},\mathtt{L},\mathtt{M})$, where:

- $\mathtt{P}$: the number of elements of a certain basic type which are presents out of the outermost compartment,
- $\mathtt{L}$: the minimum number of elements of some basic types still required by the present elements,
- $\mathtt{M}$: the maximum number of elements of basic types still tolerated by the present elements,

checking that the maximum limit is never exceed and the minimum limit is reached in every compartment.
Types are assigned to patterns and terms with the typing rules in Figure 2, where bases, assigning types to element, term and sequence variables are defined by:

$$\Delta \ ::= \ \emptyset \ \ \big| \ \ \Delta, x : (\{\mathtt{t}\}, \mathtt{D}_\mathtt{t}, \mathtt{U}_\mathtt{t}) \ \ \big| \ \ \Delta, \eta : (\mathtt{P},\mathtt{L},\mathtt{M})$$

where η denotes a sequence or term variable. A basis is *well formed* if all types in the basis are well formed.
The type of the empty sequence, (Teps) rule, has the empty multiset as *present-ms*,

$$\Delta \vdash \epsilon : (\emptyset, \emptyset, \mathscr{T}_\infty) \quad \textsf{(Teps)} \qquad \frac{a : \mathtt{t} \in \Gamma}{\Delta \vdash a : (\{\mathtt{t}\}, \mathtt{D}_\mathtt{t}, \mathtt{U}_\mathtt{t})}\ \textsf{(Tel)} \qquad \Delta, \chi : (\mathtt{P},\mathtt{L},\mathtt{M}) \vdash \chi : (\mathtt{P},\mathtt{L},\mathtt{M}) \quad \textsf{(Tvar)}$$

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

$$\frac{\Delta \vdash P : (\mathtt{P},\mathtt{L},\mathtt{M}) \quad \Delta \vdash P' : (\mathtt{P}',\mathtt{L}',\mathtt{M}') \quad (\mathtt{P},\mathtt{L},\mathtt{M}) \bowtie (\mathtt{P}',\mathtt{L}',\mathtt{M}')}{\Delta \vdash P \,|\, P' : (\mathtt{P},\mathtt{L},\mathtt{M}) \sqcup (\mathtt{P}',\mathtt{L}',\mathtt{M}')}\ \textsf{(Tpar)}$$

$$\frac{\Delta \vdash SP : (\mathtt{P},\mathtt{L},\mathtt{M}) \quad \Delta \vdash P : (\mathtt{P}',\mathtt{L}',\mathtt{M}') \quad (\mathtt{P}',\mathtt{L}',\mathtt{M}') \sqsubseteq (\mathtt{P},\mathtt{L},\mathtt{M})}{\Delta \vdash (SP)^L \rfloor P : (\mathtt{P}, \mathtt{L} \ominus \mathtt{P}', \mathtt{M} \ominus \mathtt{P}')}\ \textsf{(Tcomp)}$$

FIGURE 2. Typing Rules

because an empty sequence does not contain elements, the empty multiset as *at-least-ms* and $\mathscr{T}_\infty$ as *at-most-ms*, because the absence of elements allows the absence of limits. The type of an element, (Tel) rule, is composed by the basic type $\mathtt{t}$ as *present-ms*, and its *down-ms* and *up-ms*. The type of a variable, (Tvar) rule, is given by the basis.
The type of a sequence, a parallel composition or a looping sequence is derived from the types of the two sub-patterns. For two patterns to be combinable, the *present-ms* of one must be a subset of the *at-most-ms* of the other, i.e. the number of present elements in one must not exceed the maximum number of the same elements tolerated by the other. This key condition must hold for all sub-patterns of patterns, because the exceeding of the maximum limit for a sub-pattern makes the whole pattern not typable. On the contrary, it is not necessary to check whether all the minimum requests are reached, because this constraint depends on the elements in the whole compartment and the looping sequence containing it, if it exists: therefore this condition is checked only for a whole compartment.
Now we focus on (Tseq) and (Tpar) rules. The type of the obtained pattern is the join of the types $(\mathtt{P},\mathtt{L},\mathtt{M})$ and $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$ of the connected patterns defined as follows. The *present-ms* of the join type is the sum of the *present-ms*, $\mathtt{P} \uplus \mathtt{P}'$, i.e. the number of one element in the new type is the sum of the numbers of the same element in the old types. For getting the *at-least-ms* of the join type we remove the *present-ms* of one type from the *at-least-ms* of the other, obtaining for each type the number of elements required taking into account the presence of the basic types in the *present-ms* of the other type. The union of these multisets is the *at-least-ms* of the join type, because, as seen in the explanation of well-formed types, the maximum of two lower limits, in multiset theory calculated by the union, satisfies both lower limits. For the *at-most-ms* of the join type we do the dual, taking the intersection of the removals, i.e. their minimum. To sum up:

Definition 8 (Join of Types)**.** *Given two well formed types* $(\mathtt{P},\mathtt{L},\mathtt{M})$ *and* $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$, *we define their* join $(\mathtt{P},\mathtt{L},\mathtt{M}) \sqcup (\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *by*

$$(\mathtt{P},\mathtt{L},\mathtt{M}) \sqcup (\mathtt{P}',\mathtt{L}',\mathtt{M}') = (\mathtt{P} \uplus \mathtt{P}', (\mathtt{L} \ominus \mathtt{P}') \cup (\mathtt{L}' \ominus \mathtt{P}), (\mathtt{M} \ominus \mathtt{P}') \cap (\mathtt{M}' \ominus \mathtt{P})).$$

The type obtained by join may be not well formed, because

- its *at-least-ms* could not be a subset of its *at-most-ms*,
- there are too many present elements of a given type, i.e. the number of present elements of a given type exceed the number of tolerated elements of that type.

Since we want to restrict to well-formed types, we define compatibility between types, that impose both conditions.

Definition 9 (Type Compatibility)**.** *Two well-formed types* $(\mathtt{P},\mathtt{L},\mathtt{M})$ *and* $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *are* compatible *(written* $(\mathtt{P},\mathtt{L},\mathtt{M}) \bowtie (\mathtt{P}',\mathtt{L}',\mathtt{M}')$*) if* $(\mathtt{L}\ominus\mathtt{P}')\cup(\mathtt{L}'\ominus\mathtt{P}) \subseteq (\mathtt{M}\ominus\mathtt{P}')\cap(\mathtt{M}'\ominus\mathtt{P})$, $\mathtt{P} \subseteq \mathtt{M}'$ *and* $\mathtt{P}' \subseteq \mathtt{M}$.

Compatibility of two types is a necessary and sufficient condition to get well-formedness of the join.

Proposition 1. *Let* $(\mathtt{P},\mathtt{L},\mathtt{M})$, $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *be well-formed types:* $(\mathtt{P},\mathtt{L},\mathtt{M}) \sqcup (\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *is well-formed iff* $(\mathtt{P},\mathtt{L},\mathtt{M}) \bowtie (\mathtt{P}',\mathtt{L}',\mathtt{M}')$.

Proof. We have to show that $(\mathtt{L} \ominus \mathtt{P}') \cup (\mathtt{L}' \ominus \mathtt{P}) \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}\uplus\mathtt{P}'} \mathtt{D}_\mathtt{t}$ and $\mathtt{P} \uplus \mathtt{P}' \uplus [(\mathtt{M} \ominus \mathtt{P}') \cap (\mathtt{M}' \ominus \mathtt{P})] \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}\uplus\mathtt{P}'} (\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$.
Since $(\mathtt{P},\mathtt{L},\mathtt{M})$ and $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$ are well formed by hypothesis, we get $\mathtt{L} \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}} \mathtt{D}_\mathtt{t}$, $\mathtt{L}' \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}'} \mathtt{D}_\mathtt{t}$, $\mathtt{P} \uplus \mathtt{M} \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}} (\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$, $\mathtt{P}' \uplus \mathtt{M}' \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}'} (\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$. Then $(\mathtt{L} \ominus \mathtt{P}') \cup (\mathtt{L}' \ominus \mathtt{P}) \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}\uplus\mathtt{P}'} \mathtt{D}_\mathtt{t}$ follows from $\mathtt{L} \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}} \mathtt{D}_\mathtt{t}$ and $\mathtt{L}' \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}'} \mathtt{D}_\mathtt{t}$.
We have $\mathtt{P} \uplus \mathtt{M} = \mathtt{P} \uplus \mathtt{M} \uplus \mathtt{P}' \ominus \mathtt{P}' \supseteq \mathtt{P} \uplus \mathtt{P}' \uplus [(\mathtt{M} \ominus \mathtt{P}') \cap (\mathtt{M}' \ominus \mathtt{P})]$ since $\mathtt{P}' \subseteq \mathtt{M}$, which implies $\mathtt{P} \uplus \mathtt{P}' \uplus [(\mathtt{M} \ominus \mathtt{P}') \cap (\mathtt{M}' \ominus \mathtt{P})] \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}} (\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$ by $\mathtt{P} \uplus \mathtt{M} \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}} (\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$. Similarly we can show $\mathtt{P} \uplus \mathtt{P}' \uplus [(\mathtt{M} \ominus \mathtt{P}') \cap (\mathtt{M}' \ominus \mathtt{P})] \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}'} (\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$, so we conclude $\mathtt{P} \uplus \mathtt{P}' \uplus [(\mathtt{M} \ominus \mathtt{P}') \cap (\mathtt{M}' \ominus \mathtt{P})] \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}\uplus\mathtt{P}'} (\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$.
Note that if $(\mathtt{L} \ominus \mathtt{P}') \cup (\mathtt{L}' \ominus \mathtt{P}) \not\subseteq (\mathtt{M} \ominus \mathtt{P}') \cap (\mathtt{M}' \ominus \mathtt{P})$ the join type is clearly not well formed. If $\mathtt{P} \not\subseteq \mathtt{M}'$ it means that there are basic types $\mathtt{t} \in \mathtt{P}$, $\mathtt{t}' \in \mathtt{P}'$ such that the number of present elements of basic type $\mathtt{t}$ is bigger than the number of elements of basic type $\mathtt{t}$ allowed by the elements of basic type $\mathtt{t}'$, taking into account also the elements of basic type $\mathtt{t}$ which belongs to $\mathtt{P}'$ or possibly to an inner compartment. Therefore the join type will be not well formed. □

Finally, we consider the (Tcomp) rule. In the resulting type, the present elements are only the ones in the looping sequence, because a looping sequence makes the elements inside the compartment invisible from the outside. Since the elements in the looping sequence are influenced by the ones inside the compartment, to obtain the *at-least-ms* and *at-most-ms* of its type we must subtract the elements present in the inner pattern from the *at-least-ms* and *at-most-ms* of the looping sequence.

Definition 10 (Subtraction of Types)**.** *Given two well-formed types* $(\mathtt{P},\mathtt{L},\mathtt{M})$ *and* $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$, *we define their* subtraction *as* $(\mathtt{P},\mathtt{L} \ominus \mathtt{P}',\mathtt{M} \ominus \mathtt{P}')$.

A type obtained by subtraction is always well formed, because we are taking away the same multiset from the *at-least-ms* and *at-most-ms* of a well-formed type. Since a looping sequence encloses a compartment, we must assure that all the at-least limits in the inner compartment are reached. To do so, we require that the *present-ms* of the looping sequence satisfies all the requests of the *at-least-ms* of the inner pattern. Moreover, as in the case of compatibility we need to assure that there are not too many present basic types.

Definition 11 (Types Satisfaction)**.** *Given two well-formed types* $(\mathtt{P},\mathtt{L},\mathtt{M})$ *and* $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$, $(\mathtt{P},\mathtt{L},\mathtt{M})$ satisfies $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *(written* $(\mathtt{P}',\mathtt{L}',\mathtt{M}') \sqsubseteq (\mathtt{P},\mathtt{L},\mathtt{M})$*) if* $\mathtt{L}' \subseteq \mathtt{P}$, $\mathtt{P} \subseteq \mathtt{M}'$ *and* $\mathtt{P}' \subseteq \mathtt{M}$.

It is easy to verify that, using the typing rules in Figure 2, from the empty basis we derive a well-formed type for a term, and from a well-formed basis we derive a well-formed type for a pattern.

Reduction rules are applied only to terms such that their types are well formed and the at-least limits are reached also in the outermost compartment, i.e. their types have the empty multiset as *at-least-ms*. These terms are interesting from a biological point of view, because, since all the minimum requests are fulfilled, they represent complete systems. We call them *correct terms*:

Definition 12 (Correct Term)**.** *A term* $\vdash T : (\mathtt{P},\mathtt{L},\mathtt{M})$ *is* correct *if* $\mathtt{L} = \emptyset$.

For clarity, in the following examples, a multiset A will be denoted with the set notation by listing the types followed by their multiplicity, $\{\mathtt{t} : f_A(\mathtt{t}) \mid \mathtt{t} \in D\}$, where $\mathtt{t}_1, \mathtt{t}_2, \dots, \mathtt{t}_k : m$ means that all basic types $\mathtt{t}_1, \mathtt{t}_2, \dots, \mathtt{t}_k$ have multiplicity m. In the *at-most-ms* we write only the basic types having multiplicity 0 or finite, and we do not write the basic types having multiplicity ∞. On the contrary, in *present-ms* and *at-least-ms* we do not write the basic types having multiplicity 0. In this way we highlight only the most significant cases: in fact, an infinite multiplicity in a *at-most-ms* means no constraint, and the same for a multiplicity of zero in a *at-least-ms*.

Example 3.2. *Assuming the set of basic types*

$$\mathscr{T} = \{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c, \mathtt{t}_d\}$$

and a classification which contains

$$\{a : \mathtt{t}_a, b : \mathtt{t}_b, c : \mathtt{t}_c, d : \mathtt{t}_d\}$$

where mn, mx are:

mn	$\mathtt{t}_a$	$\mathtt{t}_b$	$\mathtt{t}_c$	$\mathtt{t}_d$
$\mathtt{t}_a$	0	0	1	0
$\mathtt{t}_b$	0	0	0	0
$\mathtt{t}_c$	2	0	0	0
$\mathtt{t}_d$	0	0	0	0

mx	$\mathtt{t}_a$	$\mathtt{t}_b$	$\mathtt{t}_c$	$\mathtt{t}_d$
$\mathtt{t}_a$	∞	∞	1	∞
$\mathtt{t}_b$	∞	∞	∞	∞
$\mathtt{t}_c$	∞	1	∞	∞
$\mathtt{t}_d$	∞	∞	∞	∞

the term

$$(A) \;\; \vdash (d)^L \rfloor (c \mid a \mid a \mid a \mid a) : (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$$

is correct, while the term

$$\vdash a \mid (d)^L \rfloor (c \mid a \mid a \mid a \mid a) : (\{\mathtt{t}_a, \mathtt{t}_d : 1\}, \{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 1\})$$

is not correct, because the term of a basic type $\mathtt{t}_a$ *requires exactly one element of basic type* $\mathtt{t}_c$*. Moreover, a term cannot have a type, if in the inner compartment containing an element of basic type* $\mathtt{t}_c$ *there are less than two elements of basic type* $\mathtt{t}_a$*, as for example*

$$a \mid (d)^L \rfloor (c \mid a).$$

This is also the case when, in the same compartment or looping sequence containing an element of basic type $\mathtt{t}_c$*, there are more than one element of basic type* $\mathtt{t}_b$*, as for example*

$$a \mid (d)^L \rfloor (c \mid a \mid a \mid b \mid b).$$

Also a term with some elements of basic type $\mathtt{t}_a$ *without any element of basic type* $\mathtt{t}_c$*, as for example*

$$a \mid (d)^L \rfloor (a \mid a \mid a)$$

does not have a type.

In the remaining of the present section we will define our typed semantics, and show that typed reductions preserve the correctness of terms.
An instantiation σ *agrees* with a basis Δ (notation $\sigma \in \Sigma_\Delta$) if $\chi : (\mathtt{P},\mathtt{L},\mathtt{M}) \in \Delta$ implies $\vdash \sigma(\chi) : (\mathtt{P},\mathtt{L},\mathtt{M})$.

The following Lemma, that can be easily proved by induction on type derivations, will be useful for the Subject Reduction Theorem:

Lemma 3.3. *If $\sigma \in \Sigma_\Delta$, then $\vdash P\sigma : (\mathtt{P}, \mathtt{L}, \mathtt{M})$ if and only if $\Delta \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.*

We are looking for a typed semantics which applied to correct terms produces only correct terms. Observe that if $Y : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash C[Y] : (\mathtt{P}', \emptyset, \mathtt{M}')$, then every term obtained filling the hole of this context with a term having type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ will be correct. This fact leads us to the following definition:

Definition 13 (Typed Hole)**.** *Given a context C and a well-formed type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$, the type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is OK for the context C if $Y : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash C[Y] : (\mathtt{P}', \emptyset, \mathtt{M}')$ for some $\mathtt{P}'$, $\mathtt{M}'$.*

For rewriting rules we are only interested in the types of the right-hand-sides, since they influence the type of the obtained term.

Definition 14 (Δ-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe Rules)**.** *A rewrite rule $P_1 \mapsto P_2$ is* Δ-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe *if $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.*

Example 3.4. *Assuming the set of basic types and the classification in Example 3.2 and the basis*

$$\Delta = \{X : (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})\}$$

the rule

$$(d)^L \rfloor (b \,|\, X) \mapsto b \,|\, (d)^L \rfloor X$$

is a Δ-$(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$-safe rule. In fact we derive

$$\Delta \vdash b \,|\, (d)^L \rfloor X : (\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty).$$

Using Definitions 13 and 14, if we apply a rule whose right-hand-side has type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$, and this type is *OK* for the context, we obtain a correct term.

Definition 15 (Typed Semantics)**.** *Given a finite set of rewriting rules $\mathcal{R}$, the* typed semantics *of* CLS *is the least relation closed with respect to $\equiv$ and satisfying the following sets of rules:*

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

As expected, typed reduction preserve correctness.

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

Proof. From Definition 15, we have that T' is $C[P_2\sigma]$, and, from Definition 14, we have that $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{L}, \mathtt{M})$ for some Δ, $\mathtt{P}$, $\mathtt{L}$, $\mathtt{M}$. Lemma 3.3 and $\sigma \in \Sigma_\Delta$ imply that $\vdash P_2\sigma : (\mathtt{P}, \mathtt{L}, \mathtt{M})$. Since, from Definition 15, the type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is *OK* for C, we conclude that $\vdash C[P_2\sigma] : (\mathtt{P}', \emptyset, \mathtt{M}')$ for some $\mathtt{P}'$, $\mathtt{M}'$. □

	P_1	P_2	X	C
(A)	a	b	—	$(d)^L \rfloor (c \,\vert\, a \,\vert\, a \,\vert\, a \,\vert\, \Box)$
(B)	$(d)^L \rfloor (b \,\vert\, X)$	$b \,\vert\, (d)^L \rfloor X$	$c \,\vert\, a \,\vert\, a \,\vert\, a$	$\Box$
(C)	a	b	—	$b \,\vert\, (d)^L \rfloor (c \,\vert\, a \,\vert\, a \,\vert\, \Box)$
(D)	$(d)^L \rfloor (b \,\vert\, X)$	$b \,\vert\, (d)^L \rfloor X$	$c \,\vert\, a \,\vert\, a$	$b \,\vert\, \Box$

FIGURE 3. Rules, Instantiations and Contexts of Example 3.6

	Δ	type of $\mathtt{P}_2\sigma$	type of $C[\mathtt{P}_2\sigma]$
(A)	—	$(\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty)$	$(\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$
(B)	$X : (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$	$(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$	$(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$
(C)	—	$(\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty)$	$(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$
(D)	$X : (\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$	$(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$	$(\{\mathtt{t}_b : 2, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$

FIGURE 4. Basis and Typings of Example 3.6

Example 3.6. *We study the behavior of the term (A) in Example 3.2 using the rules*

$$(1)\ a \mapsto b \quad \text{and} \quad (2)\ (d)^L \rfloor (b \,\vert\, X) \mapsto b \,\vert\, (d)^L \rfloor X.$$

Rules, instantiations and contexts for the next reductions are reported in Figure 3, and their basis and typings are reported in Figure 4.
On term (A) of Example 3.2, we can only apply rule (1), obtaining the correct term

$$(B)\ \vdash (d)^L \rfloor (c \,\vert\, a \,\vert\, a \,\vert\, a \,\vert\, b) : (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty).$$

On term (B), we cannot apply rule (1), because the basic type of c allows no more than one element having basic type $\mathtt{t}_b$, and the term derived from the application of this rule would not respect this constraint. We can only apply rule (2), obtaining the correct term

$$(C)\ \vdash b \,\vert\, (d)^L \rfloor (c \,\vert\, a \,\vert\, a \,\vert\, a) : (\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty).$$

On term (C) , we can only apply rule (1), obtaining the correct term

$$(D)\ \vdash b \,\vert\, (d)^L \rfloor (c \,\vert\, a \,\vert\, a \,\vert\, b) : (\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty).$$

For the same reason as for the term (B), on the term (D) we can apply only rule (2), obtaining the correct term

$$(E)\ \vdash b \,\vert\, b \,\vert\, (d)^L \rfloor (c \,\vert\, a \,\vert\, a) : (\{\mathtt{t}_b : 2, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty).$$

Rule (1) cannot be applied, because the basic type of c needs at least two elements having basic type $\mathtt{t}_a$, and the term derived from the application of this rule does not respect this constraint. Since also rule (2) cannot be applied, the term (E) cannot reduce using rules (1) and (2).

We end this section by stating two lemmas on properties of typing rules which will be used to show the theorems of next section.

Lemma 3.7 (Weakening)**.** *If $\Delta \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$ and $\Delta \subseteq \Delta'$, then $\Delta' \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.*

Proof. By induction on derivations. □

Lemma 3.8. *If* $\Delta \vdash C[P] : (\mathtt{P}, \mathtt{L}, \mathtt{M})$ *then*

(1) $\Delta \vdash P : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$ *for some* $(\mathtt{P}', \mathtt{L}', \mathtt{M}')$*, and*

(2) *if* P' *is such that* $\Delta \vdash P' : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$*, then* $\Delta \vdash C[P'] : (\mathtt{P}, \mathtt{L}, \mathtt{M})$*.*

Proof. By induction on contexts. □

4. Inference

We use the machinery of principal typing [17] to infer the OK relation between types and contexts and which rules are Δ-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe. From the term we want to reduce we obtain a set of constraints: if they are fulfilled by a rule, then the rule can be applied to the term preserving correctness. In this way, we can decide the applicability of the rules.
We convene that for each element variable x there is an *e-type* variable η_x ranging over basic types, and for each term or sequence variable ψ there are three variables π_ψ, λ_ψ, μ_ψ (called *p-type* variable, *l-type* variable and *m-type* variable) ranging over multisets of basic types. Moreover we convene that Π ranges over formal unions of multisets of basic types, *e-type* variables and *p-type* variables, Λ ranges over unions of multisets of basic types and *l-type* variables, and Ω ranges over unions of multisets of basic types and *m-type* variables. We denote by δ a generic *p-type*, *l-type*, *m-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 triples of their *p-type* variables, *l-type* variables and *m-type* variables:

$$\Theta \ ::= \ \emptyset \ \mid \ \Theta, x : \eta_x \ \mid \ \Theta, \psi : (\pi_\psi, \lambda_\psi, \mu_\psi).$$

The rules for inferring principal typings use judgments of the shape:

$$\vdash P : \Theta; (\Pi, \Lambda, \Omega); \Xi$$

where Θ is the *principal basis* in which P is well formed, (Π, Λ, Ω) is the *principal type* of P, and Ξ is the set of constraints that should be satisfied. Figure 5 gives these inference rules, derived from the typing rules in Figure 2.

Rules (Reps) and (Rel) directly derive from rules (Teps) and (Tel). The rules for typing variables (rules (Rvar_1) and (Rvar_2)) put the variable with its type in the basis. In rules (Rseq), (Rpar) and (Rcomp), the principal type is derived as in (Tseq), (Tpar) and (Tcomp) rules respectively. The set of constraints is the union between the constraints in the premise of the rule itself and the constraints in the premise of (Tseq), (Tpar) and (Tcomp) rules, respectively. The principal basis is the union of the principal bases of the composing patterns, without renaming, because each variable ψ or x is associated to an unique triple of *p-type*, *l-type*, *m-type* variables or to an unique *e-type* variable, respectively.
The key difference between inference rules, in Figure 5, and typing rules, in Figure 2, is that the conditions of type compatibility and type satisfaction are not premises, but conclusions. In this way, at the end of inference all these conditions

$$\vdash \epsilon : \emptyset; (\emptyset, \emptyset, \infty); \emptyset \quad (\mathsf{Reps}) \quad \vdash x : \{x : (\{\eta_x\}, \mathtt{D}_{\eta_x}, \mathtt{U}_{\eta_x})\}; (\{\eta_x\}, \mathtt{D}_{\eta_x}, \mathtt{U}_{\eta_x}); \emptyset \quad (\mathsf{Rvar}_1)$$

$$\vdash \psi : \{\psi : (\pi_\psi, \lambda_\psi, \mu_\psi)\}; (\pi_\psi, \lambda_\psi, \mu_\psi); \emptyset \quad (\mathsf{Rvar}_2) \qquad \frac{a : \mathtt{t} \in \Gamma}{\vdash a : \emptyset; (\{\mathtt{t}\}, \mathtt{D}_\mathtt{t}, \mathtt{U}_\mathtt{t}); \emptyset}\ (\mathsf{Rel})$$

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

$$\frac{\vdash P : \Theta; (\Pi, \Lambda, \Omega); \Xi \qquad \vdash P' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'}{\vdash P \,|\, P' : \Theta \cup \Theta'; (\Pi, \Lambda, \Omega) \sqcup (\Pi', \Lambda', \Omega'); \Xi \cup \Xi' \cup \{(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')\}}\ (\mathsf{Rpar})$$

$$\frac{\vdash SP : \Theta; (\Pi, \Lambda, \Omega); \Xi \qquad \vdash P' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'}{\vdash (SP)^L \rfloor P : \Theta \cup \Theta'; (\Pi, \Lambda \ominus \Pi', \Omega \ominus \Pi'); \Xi \cup \Xi' \cup \{(\Pi', \Lambda', \Omega') \sqsubseteq (\Pi, \Lambda, \Omega)\}}\ (\mathsf{Rcomp})$$

FIGURE 5. Inference Rules

create a set of constraints, that must be checked to decide the applicability of the rules.

Example 4.1. *We can use the inference rules in Figure 5 to infer the types of the right-side patterns of the rules in Example 3.6, where, again, we assume the set of basic types and the classification of Example 3.2, obtaining*

$$\begin{array}{l} \vdash b : \emptyset; (\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty); \emptyset \\ \vdash b \,|\, (d)^L \rfloor X : \Theta; (\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty) \sqcup (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty); \Xi \end{array}$$

where

$$\begin{array}{rcl} \Theta & = & \{\ X : (\pi_X, \lambda_X, \mu_X)\ \} \\ \Xi & = & \{\ \ (\pi_X, \lambda_X, \mu_X) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty), \\ & & (\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty) \bowtie (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)\ \ \} \end{array}$$

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, *l-type* variables and *m-type* variables to multisets 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* $\vdash P : \Theta; (\Pi, \Lambda, \Omega); \Xi$ *and* m *is a type mapping which satisfies* Ξ*, then* $\mathsf{m}(\Theta) \vdash P : (\mathsf{m}(\Pi), \mathsf{m}(\Lambda), \mathsf{m}(\Omega))$.

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
  $\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Pi, \Lambda, \Omega) \sqcup (\Pi', \Lambda', \Omega'); \Xi \cup \Xi' \cup \{(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')\}$
  and the assumptions are

$$\vdash SP : \Theta; (\Pi, \Lambda, \Omega); \Xi \text{ and } \vdash SP' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'.$$

Since m satisfies Ξ and Ξ', by induction hypothesis, and weakening (Lemma 3.7), we derive

$$\mathsf{m}(\Theta \cup \Theta') \vdash SP : (\mathsf{m}(\Pi), \mathsf{m}(\Lambda), \mathsf{m}(\Omega))$$
$$\mathsf{m}(\Theta \cup \Theta') \vdash SP' : (\mathsf{m}(\Pi'), \mathsf{m}(\Lambda'), \mathsf{m}(\Omega')).$$

Moreover, since m satisfies $(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')$, we have

$$(\mathsf{m}(\Pi), \mathsf{m}(\Lambda), \mathsf{m}(\Omega)) \bowtie (\mathsf{m}(\Pi'), \mathsf{m}(\Lambda'), \mathsf{m}(\Omega')).$$

Therefore the rule (Tseq) can be applied, and

$$\mathsf{m}(\Theta \cup \Theta') \vdash SP{\cdot}SP' : (\mathsf{m}(\Pi), \mathsf{m}(\Lambda), \mathsf{m}(\Omega)) \sqcup (\mathsf{m}(\Pi'), \mathsf{m}(\Lambda'), \mathsf{m}(\Omega')).$$

- For rules (Rpar), and (Rcomp) the result can be proved like for rule (Rseq).

□

Theorem 4.3 (Completeness of Type Inference). *If* $\Delta \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$*, then* $\vdash P : \Theta; (\Pi, \Lambda, \Omega); \Xi$ *for some* Θ, Π, Λ, Ω, Ξ *and there is a type mapping* m *that satisfies* Ξ *and such that* $\Delta \supseteq \mathsf{m}(\Theta)$, $\mathtt{P} = \mathsf{m}(\Pi)$, $\mathtt{L} = \mathsf{m}(\Lambda)$, $\mathtt{M} = \mathsf{m}(\Omega)$.

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

- 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
$$\Delta \vdash SP{\cdot}SP' : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \sqcup (\mathtt{P}', \mathtt{L}', \mathtt{M}'),$$
and the assumptions are
$$\Delta \vdash SP : (\mathtt{P}, \mathtt{L}, \mathtt{M}),\ \Delta \vdash SP' : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$$
and the condition $(\mathtt{P}', \mathtt{L}', \mathtt{M}') \bowtie (\mathtt{P}', \mathtt{L}', \mathtt{M}')$. By induction hypothesis, there are Θ, Π, Λ, Ω, Ξ, Θ', Π', Λ', Ω', Ξ' such that
$$\vdash SP : \Theta; (\Pi, \Lambda, \Omega); \Xi \text{ and } \vdash SP' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'.$$
These are the assumptions of rule (Rseq), whose conclusion is
$$\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Pi, \Lambda, \Omega) \sqcup (\Pi', \Lambda', \Omega'); \Xi \cup \Xi' \cup \{(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')\}.$$
Moreover, by induction there is a type mapping m' satisfying Ξ such that $\Delta \supseteq \mathsf{m}'(\Theta)$, $\mathtt{P} = \mathsf{m}'(\Pi)$, $\mathtt{L} = \mathsf{m}'(\Lambda)$ and $\mathtt{M} = \mathsf{m}'(\Omega)$, and there is a type mapping m'' satisfying Ξ' such that $\Delta \supseteq \mathsf{m}''(\Theta')$, $\mathtt{P}' = \mathsf{m}''(\Pi')$, $\mathtt{L}' = \mathsf{m}''(\Lambda')$ and $\mathtt{M}' = \mathsf{m}''(\Omega')$. Therefore, we derive $\Delta \supseteq \mathsf{m}'(\Theta) \cup \mathsf{m}''(\Theta')$ and $(\mathtt{P}, \mathtt{L}, \mathtt{M}) \sqcup (\mathtt{P}', \mathtt{L}', \mathtt{M}') = (\mathsf{m}'(\Pi), \mathsf{m}'(\Lambda), \mathsf{m}'(\Omega)) \sqcup (\mathsf{m}''(\Pi'), \mathsf{m}''(\Lambda'), \mathsf{m}''(\Omega'))$. Since the bases $\mathsf{m}'(\Theta)$ and $\mathsf{m}''(\Theta')$ are both subsets of the same basis Δ, then for all the (*e-type*, *p-type*, *l-type* or *m-type*) variables δ such that $\delta \in dom(\mathsf{m}') \cap dom(\mathsf{m}'')$ we get $\mathsf{m}'(\delta) = \mathsf{m}''(\delta)$. Therefore the mapping m
$$\mathsf{m}(\delta) = \begin{cases} \mathsf{m}'(\delta) & \text{if } \delta \in dom(\mathsf{m}') \\ \mathsf{m}''(\delta) & \text{if } \delta \in dom(\mathsf{m}'') \end{cases}$$
is well defined.
Moreover, since m satisfies Ξ, Ξ' and $(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')$, 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.

□

We use the inference rules to decide applicability of typed reduction rules for Δ-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe rules. The first step is to understand when a type mapping makes a rule a Δ-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe rule, i.e. when it satisfies the constraints in Definition 14.

Lemma 4.4 (Characterization of Δ-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe rules)**.** *A rule* $P_1 \mapsto P_2$ *is* Δ-$(\mathtt{P},\mathtt{L},\mathtt{M})$*-safe if and only if the type mapping* m *defined by the basis* Δ*, i.e. such that*

- $\mathsf{m}(\eta_x) = \mathtt{t}$ *if* $\Delta(x) = \{\mathtt{t}\}$
- $\mathsf{m}(\pi_\psi) = \mathtt{P}'$ *if* $\Delta(\psi) = (\mathtt{P}',\mathtt{L}',\mathtt{M}')$
- $\mathsf{m}(\lambda_\psi) = \mathtt{L}'$ *if* $\Delta(\psi) = (\mathtt{P}',\mathtt{L}',\mathtt{M}')$
- $\mathsf{m}(\mu_\psi) = \mathtt{M}'$ *if* $\Delta(\psi) = (\mathtt{P}',\mathtt{L}',\mathtt{M}')$

satisfies the set of constraints $\Xi_2 \cup \{\Pi_2 = \mathtt{P}\} \cup \{\Lambda_2 = \mathtt{L}\} \cup \{\Omega_2 = \mathtt{M}\}$*, where* $\vdash P_2 : \Theta_2; (\Pi_2, \Lambda_2, \Omega_2); \Xi_2$*.*

We can apply Δ-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe rules only in contexts in which the type $(\mathtt{P},\mathtt{L},\mathtt{M})$ is *OK*, so we must characterize also the *OK* relation. To check this relation it is not necessary to consider the whole context, but only the part of the context influenced by the typing of the hole: from the typing rules we can see that the typing of a term inside two nested looping sequences does not influence the typing of the terms outside the outermost looping sequence. We call *core of the context* the subcontext including the hole and the part of the context influenced by the type of the hole. The following definition formalizes this notion.

Definition 16. *The* core of the context C *(notation* $\mathsf{core}(C)$*) is defined by:*

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

Thanks to this notion, we can characterize the *OK* relation using a small number of constraints.

Lemma 4.5 (Characterization of *OK* Relation)**.** *Let the context* C *be such that* $\vdash C[T] : (\mathtt{P}_0, \emptyset, \mathtt{M}_0)$ *for some* T*,* $\mathtt{P}_0$*,* $\mathtt{M}_0$*. A type* $(\mathtt{P},\mathtt{L},\mathtt{M})$ *is OK for* C *if and only if the type mapping* m *defined by*

- $\mathsf{m}(\pi_Y) = \mathtt{P}$*,*
- $\mathsf{m}(\lambda_Y) = \mathtt{L}$*,*
- $\mathsf{m}(\mu_Y) = \mathtt{M}$*,*

satisfies the set of constraints

$$\Xi \cup \{\Lambda = \emptyset \textit{ if } \mathsf{core}(C) = C\},$$

where $\vdash \mathsf{core}(C)[Y] : \{Y : (\pi_Y, \lambda_Y, \mu_Y)\}; (\Pi, \Lambda, \Omega); \Xi$*.*

Example 4.6. *Using Lemma 4.5, the constraints making OK the type associated with a generic variable* Y *for the contexts in Example 3.6 are:*

(A) $(\pi_Y, \lambda_Y, \mu_Y) \bowtie (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$
$((\pi_Y, \lambda_Y, \mu_Y) \sqcup (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty) \quad \emptyset = \emptyset$
(B) $\lambda_Y = \emptyset$
(C) $(\pi_Y, \lambda_Y, \mu_Y) \bowtie (\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$
$((\pi_Y, \lambda_Y, \mu_Y) \sqcup (\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty) \quad \emptyset = \emptyset$
(D) $(\pi_Y, \lambda_Y, \mu_Y) \bowtie (\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty) \quad \lambda_Y \ominus \{\mathtt{t}_b : 1\} = \emptyset$

Note that Λ is $\emptyset$ in (A) and (C).

Once we have characterized the Δ-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe rules, and also the OK relation, we can infer the applicability of a rewrite rule by checking if the type mapping respects the constraints derived for these rules.

Theorem 4.7 (Applicability of rewrite rules)**.** *Let*

$$\vdash P_2 : \Theta_2; (\Pi_2, \Lambda_2, \Omega_2); \Xi_2 \quad , \ \vdash \mathsf{core}(C)[Y] : \{Y : (\pi_Y, \lambda_Y, \mu_Y)\}; (\Pi_C, \Lambda_C, \Omega_C); \Xi_C$$

and $P_1\sigma \not\equiv \epsilon$. Then the rule $P_1 \mapsto P_2$ can be applied to the term $C[P_1\sigma]$ such that $\vdash C[P_1\sigma] : (\mathtt{P}_0, \emptyset, \mathtt{M}_0)$ (for some $\mathtt{P}_0$, $\mathtt{M}_0$) if and only if the type mapping m defined by

- $\mathsf{m}(\eta_x) = \mathtt{t}$ *if* $\sigma(x) : \mathtt{t} \in \Gamma$,
- $\mathsf{m}(\pi_\psi) = \mathtt{P}'$ *if* $\vdash \sigma(\psi) : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$,
- $\mathsf{m}(\lambda_\psi) = \mathtt{L}'$ *if* $\vdash \sigma(\psi) : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$,
- $\mathsf{m}(\mu_\psi) = \mathtt{M}'$ *if* $\vdash \sigma(\psi) : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$,

satisfies the following sets of constraints:

$$\Xi_2 \cup \Xi_C \cup \{(\pi_Y = \Pi_2), (\lambda_Y = \Lambda_2), (\mu_Y = \Omega_2)\} \cup \{\Lambda_C = \emptyset \textit{ if } \mathsf{core}(C) = C\}$$

Proof. We define the basis Δ as follows:

- $x : (\{\mathtt{t}\}, \mathtt{D_t}, \mathtt{U_t}) \in \Delta$ if $\sigma(x) : \mathtt{t} \in \Gamma$, and
- $\psi : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \in \Delta$ if $\vdash \sigma(\psi) : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$.

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

- $\mathsf{m}(\eta_x) = \mathtt{t}$ iff $x : (\{\mathtt{t}\}, \mathtt{D_t}, \mathtt{U_t}) \in \Delta$
- $\mathsf{m}(\pi_\psi) = \mathtt{P}'$ iff $\psi : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \in \Delta$
- $\mathsf{m}(\lambda_\psi) = \mathtt{L}'$ iff $\psi : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \in \Delta$
- $\mathsf{m}(\mu_\psi) = \mathtt{M}'$ iff $\psi : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \in \Delta$.

($\Leftarrow$)**:** If the mapping m satisfies the set of constraints $\Xi_2 \cup \Xi_C \cup \{(\pi_Y = \Pi_2), (\lambda_Y = \Lambda_2), (\mu_Y = \Omega_2)\} \cup \{\Lambda_C = \emptyset$ if $\mathsf{core}(C) = C\}$, then $\mathsf{m}(\pi_Y) = \mathsf{m}(\Pi_2) = \mathtt{P}, \mathsf{m}(\lambda_Y) = \mathsf{m}(\Lambda_2) = \mathtt{L}, \mathsf{m}(\mu_Y) = \mathsf{m}(\Omega_2) = \mathtt{M}$ for some $\mathtt{P}$, $\mathtt{L}$, $\mathtt{M}$, and by Lemma 4.5 the context C is OK for $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ and by Lemma 4.4 the rule $P_1 \mapsto P_2$ is Δ-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe; we get $C[P_1\sigma] \Longrightarrow C[P_2\sigma]$.

($\Rightarrow$)**:** If $C[P_1\sigma] \Longrightarrow C[P_2\sigma]$, then the rule $P_1 \mapsto P_2$ is Δ-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe for some $\mathtt{P}$, $\mathtt{L}$, $\mathtt{M}$, and the context C is OK for $(\mathtt{P}, \mathtt{L}, \mathtt{M})$. By Lemmas 4.5 and 4.4, $\mathsf{m}(\pi_Y) = \mathsf{m}(\Pi_2) = \mathtt{P}, \mathsf{m}(\lambda_Y) = \mathsf{m}(\Lambda_2) = \mathtt{L}, \mathsf{m}(\mu_Y) = \mathsf{m}(\Omega_2) = \mathtt{M}$, and the mapping m satisfies the set of constraints $\Xi_2 \cup \Xi_C \cup \{(\pi_Y = \Pi_2), (\lambda_Y = \Lambda_2), (\mu_Y = \Omega_2)\} \cup \{\Lambda_C = \emptyset$ if $\mathsf{core}(C) = C\}$.

□

Example 4.8. *We use the Theorem 4.7 on the terms of Example 3.6. Each type mapping derived from the instantiation and from the constraints* $\{(\pi_Y = \Pi_2), (\lambda_Y = \Lambda_2), (\mu_Y = \Omega_2)\}$*, reported in Fig. 6, satisfies its own set of constraints for the right-hand of the rules and* OK *relations for the contexts, reported in Examples 4.1 and 4.6, respectively.*

(*A*) *rule constraints:* $\emptyset$
context constraints: $(\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty) \bowtie (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$
$(\{\mathtt{t}_a : 3, \mathtt{t}_b, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b, \mathtt{t}_c : 0\}) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty) \quad \emptyset = \emptyset$

(*B*) *rule constraints:* $(\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\}) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$
$(\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty) \bowtie (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$
context constraints: $\emptyset = \emptyset$

(*C*) *rule constraints:* $\emptyset$
context constraints: $(\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty) \bowtie (\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$
$(\{\mathtt{t}_a : 2, \mathtt{t}_b, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b, \mathtt{t}_c : 0\}) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty) \quad \emptyset = \emptyset$

(*D*) *rule constraints:* $(\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\}) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$
$(\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty) \bowtie (\{\mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty)$
context constraints: $(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \mathscr{T}_\infty) \bowtie (\{\mathtt{t}_b : 1\}, \emptyset, \mathscr{T}_\infty) \quad \emptyset = \emptyset$

	π_X	λ_X	μ_X	π_Y	λ_Y	μ_Y
(A)	—	—	—	$\mathtt{t}_b : 1$	$\emptyset$	$\mathscr{T}_\infty$
(B)	$\mathtt{t}_a : 3, \mathtt{t}_c : 1$	$\emptyset$	$\mathtt{t}_b : 1, \mathtt{t}_c : 0$	$\mathtt{t}_b, \mathtt{t}_d : 1$	$\emptyset$	$\mathscr{T}_\infty$
(C)	—	—	—	$\mathtt{t}_b : 1$	$\emptyset$	$\mathscr{T}_\infty$
(D)	$\mathtt{t}_a : 2, \mathtt{t}_c : 1$	$\emptyset$	$\mathtt{t}_b : 1, \mathtt{t}_c : 0$	$\mathtt{t}_b, \mathtt{t}_d : 1$	$\emptyset$	$\mathscr{T}_\infty$

FIGURE 6. Type mappings of Example 4.8

5. EXAMPLE

The type discipline presented in Section 3 can be used both to describe the structure of an element and to limit the presence of some elements. We present an example for each use: for the structure description we present the hemoglobin variants, and for the limitation we present the regulation between cell death and division, an example of homeostatic balance in living organisms. The aim of these examples is not to describe a complete biological case study, but to give an idea of the possible uses of the type discipline.

5.1. HEMOGLOBIN VARIANTS

Hemoglobin (abbreviated Hb) is the oxygen-transport metalloprotein in the red blood cells of vertebrates, and in the tissues of some invertebrates. It consists mostly of proteins (the globin chains), which usually differ between species, and even within a species, although one sequence is usually a most common one in each species. In humans, the hemoglobin molecule is an assembly of four globular protein subunits: the most common, with a normal amount over 95%, is the hemoglobin A, consisting of two α and two β subunits, but there are some hemoglobin variants, as reported in Table 1. Many of these cause no disease, but some of these cause a group of hereditary diseases, known as hemoglobinopathies: the most known are sickle-cell disease, in which red blood cells assume an abnormal and rigid shape, and thalassemias, that usually result in underproduction of normal globin proteins.

Name	Structure	Informations
A	$\alpha_2\beta_2$	The most common
A_2	$\alpha_2\delta_2$	It has a normal range of 1.5-3.5%
F	$\alpha_2\gamma_2$	The one presents in the fetus
H	β_4	It may be present in variants of α thalassemia
Barts	δ_4	It may be present in variants of α thalassemia

TABLE 1. Some hemoglobin variants in humans.

We want to model the different kinds of hemoglobin: we model an hemoglobin protein as a looping sequence having the element h on the surface and containing, depending on the kind of hemoglobin, four subunits, chosen between α, β, γ and δ, and one of the elements A, A_2, F, H or B, representing the different kinds of hemoglobin in Table 1. For example, the term modeling the hemoglobin A is $(h)^L \rfloor (A \,|\, \alpha{\cdot}\alpha{\cdot}\beta{\cdot}\beta)$, and the one for hemoglobin H is $(h)^L \rfloor (H \,|\, \beta{\cdot}\beta{\cdot}\beta{\cdot}\beta)$. According to the structure of each hemoglobin variant, we create the basic types shown in Table 2.

element	basic type	minimum & maximum
α	$\mathtt{t}_\alpha$	—
β	$\mathtt{t}_\beta$	—
γ	$\mathtt{t}_\gamma$	—
δ	$\mathtt{t}_\delta$	—
A	$\mathtt{t}_A$	$\mathtt{t}_\gamma, \mathtt{t}_\delta, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\alpha, \mathtt{t}_\beta : 2$
A_2	$\mathtt{t}_{A_2}$	$\mathtt{t}_\beta, \mathtt{t}_\gamma, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\alpha, \mathtt{t}_\delta : 2$
F	$\mathtt{t}_F$	$\mathtt{t}_\beta, \mathtt{t}_\delta, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\alpha, \mathtt{t}_\gamma : 2$
H	$\mathtt{t}_H$	$\mathtt{t}_\alpha, \mathtt{t}_\gamma, \mathtt{t}_\delta, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\beta : 4$
Barts	$\mathtt{t}_B$	$\mathtt{t}_\alpha, \mathtt{t}_\beta, \mathtt{t}_\gamma, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\delta : 4$

TABLE 2. Basic types for hemoglobin variants and subunits.

Using the typed extension of CLS and these basic type, no rule can change the structure of the different kinds of hemoglobin without removing or modifying its structural element A, A_2, F, H or B.

5.2. Cell Death and Division

In multicellular organisms the life of the cells is ruled by two oppose processes: a cell division process, resulting in cell multiplication (mitosis for eukaryotic cells and binary fission for prokaryotic cells), and a process of programmed cell death, called apoptosis. In an adult organism, the rate of these processes must be balanced: an excess of cell death leads to cell loss, and an excess of cell division leads to tumors. For this reason, the number of cells is kept relatively constant through cell death and division. Using our type discipline, we can model this behavior in a simple way. We model a cell as a looping sequence having the element c on the surface and containing the other elements of interest for the study, in this case only the element a: $(c)^L \rfloor a$. We assume the organism can support from 2 to 8 cells, and the element a does not have any request. Assuming the set of basic types $\mathscr{T} = \{\mathtt{t}_a, \mathtt{t}_c\}$, a basic type $\mathtt{t}_a$ for the element a and a basic type $\mathtt{t}_c$ for the element c, the functions mn and mx that model this behavior are:

mn	$\mathtt{t}_a$	$\mathtt{t}_c$
$\mathtt{t}_a$	0	0
$\mathtt{t}_c$	0	1

mx	$\mathtt{t}_a$	$\mathtt{t}_c$
$\mathtt{t}_a$	$\mathscr{T}_\infty$	$\mathscr{T}_\infty$
$\mathtt{t}_c$	$\mathscr{T}_\infty$	7

In this model, the rules for cell death and cell division are

$$\begin{array}{ll} (de) & (\widetilde{x}\cdot c)^L \rfloor X \mapsto \epsilon \\ (di) & (\widetilde{x}\cdot c)^L \rfloor X \mapsto (\widetilde{x}\cdot c)^L \rfloor X \mid (\widetilde{x}\cdot c)^L \rfloor X \end{array}$$

respectively. For the right-side patterns of these rules we can infer the following types:

$$(rp1)\ \vdash \epsilon : \emptyset; (\emptyset, \emptyset, \mathscr{T}_\infty); \emptyset$$

$$(rp2)\ \vdash (\widetilde{x}\cdot c)^L \rfloor X \mid (\widetilde{x}\cdot c)^L \rfloor X : \Theta_{\widetilde{x},X}; (\Pi,\Lambda,\Omega) \sqcup (\Pi,\Lambda,\Omega); \Xi \cup \{(\Pi,\Lambda,\Omega) \bowtie (\Pi,\Lambda,\Omega)\}$$

where

$$\begin{array}{lcl} \Theta_{\widetilde{x},X} & = & \{\widetilde{x} : (\pi_{\widetilde{x}}, \lambda_{\widetilde{x}}, \mu_{\widetilde{x}}), X : (\pi_X, \lambda_X, \mu_X)\} \\ \Pi & = & \{\mathtt{t}_c : 1\} \uplus \pi_{\widetilde{x}} \\ \Lambda & = & ((\{\mathtt{t}_c : 1\} \ominus \pi_{\widetilde{x}}) \cup (\lambda_{\widetilde{x}} \ominus \{\mathtt{t}_c : 1\})) \ominus \pi_X \\ \Omega & = & ((\{\mathtt{t}_c : 7\} \ominus \pi_{\widetilde{x}}) \cap (\mu_{\widetilde{x}} \ominus \{\mathtt{t}_c : 1\})) \ominus \pi_X \\ \Xi & = & \{(\{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 7\}) \bowtie (\pi_{\widetilde{x}}, \lambda_{\widetilde{x}}, \mu_{\widetilde{x}}) \quad \cup \\ & & (\pi_X, \lambda_X, \mu_X)\} \sqsubseteq ((\{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 7\}) \sqcup (\pi_{\widetilde{x}}, \lambda_{\widetilde{x}}, \mu_{\widetilde{x}})). \end{array}$$

We want to study the behavior of a system composed by two cells, $(c)^L \rfloor a \mid (c)^L \rfloor a$, and we try to apply rules (de) and (di) on it. Because both rules have the same left side pattern, they use the same instantiation, $\sigma(\widetilde{x}) = \epsilon$ and $\sigma(X) = a$, and also the same context $(c)^L \rfloor a \mid \square$, for which we derive, using the rules in Figure 5, the inferred type:

$$\vdash (c)^L \rfloor a \mid Y : \Theta_Y; (\Pi_C, \Lambda_C, \Omega_C); \Xi_C$$

where

$$\begin{array}{lcl} \Theta_Y & = & \{Y : (\pi_Y, \lambda_Y, \mu_Y)\} \\ \Pi_C & = & \{\mathtt{t}_c : 1\} \uplus \pi_Y \\ \Lambda_C & = & (\{\mathtt{t}_c : 1\} \ominus \pi_Y) \cup (\lambda_Y \ominus \{\mathtt{t}_c : 1\}) \\ \Omega_C & = & (\{\mathtt{t}_c : 7\} \ominus \pi_Y) \cap (\mu_Y \ominus \{\mathtt{t}_c : 1\}) \\ \Xi_C & = & \{(\{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 7\}) \bowtie (\pi_Y, \lambda_Y, \mu_Y). \end{array}$$

We use Theorem 4.7 to check the applicability of the rules. First of all, from the instantiation σ we get the type mapping

$\pi_{\tilde{x}}$	$\lambda_{\tilde{x}}$	$\mu_{\tilde{x}}$	π_X	λ_X	μ_X
$\emptyset$	$\emptyset$	$\mathscr{T}_\infty$	$\mathtt{t}_a : 1$	$\emptyset$	$\mathscr{T}_\infty$

and we use it to instantiate the type and the constraints of the patterns $(rp1)$ and $(rp2)$, obtaining that the constraints of both patterns are satisfied, and

$$\Pi = \{\mathtt{t}_c : 1\} \qquad \Lambda = \{\mathtt{t}_c : 1\} \qquad \Omega = \{\mathtt{t}_c : 7\}.$$

Now we check the other constraints in Theorem 4.7. For the rule (di) we have

$$\pi_Y = \{\mathtt{t}_c : 2\} \qquad \lambda_Y = \emptyset \qquad \mu_Y = \{\mathtt{t}_c : 6\}.$$

The constraints of the context, Ξ_C, and the constraint $\Lambda_C = \emptyset$ are satisfied, then we can apply the rule.
For the rule (de) we have

$$\pi_Y = \emptyset \qquad \lambda_Y = \emptyset \qquad \mu_Y = \mathscr{T}_\infty.$$

The constraints of the context, Ξ_C, are satisfied, but the constraint $\{\mathtt{t}_c : 1\} = \Lambda_C = \emptyset$ is not satisfied, then we cannot apply the rule: in fact, this rule kills a cell, an invalid behavior in an organism composed by only 2 cells, the minimum required number. In a dual way, cell division is not possible in an organism composed by 8 cells.

6. Conclusions

In this paper we introduced a type discipline for the Calculus of the Looping Sequences which allows to describe and to limit the structure of systems and subsystems: this behavior cannot be easily reproduced only by means of reduction rules. In this way we may transfer the complexity of some biological properties from reduction rules to types, describing the behavior of biological systems using only general rules.
In [6, 9] some rules are classified as Δ-safe rules, i.e. rules having the same type for the left-side and the right-side patterns: these rules do not change the type of terms to which they are applied. In the present type discipline, for typing we count the number of elements of a term. Since rules usually change something in the term, adding or removing elements, in a rule the number of elements in the right-hand-side is different from the number of elements in the left-hand-side, and therefore their types are also different. According to this idea, we decided to not include Δ-safe rules in the present semantics, because using the present type discipline very few reduction rules are Δ-safe.
In nature, the minimum and maximum levels presented in this paper can be sometimes exceeded, even if this would lead the system to disease: even if undesiderable, the balance between cell death and division can be broken, leading to death or tumors. On the contrary, our typed semantics completely exclude these kinds of situations. According to this idea, we could modify our typed semantics, allowing

transitions which lead to untypable terms, but signaling that some undesired state has been reached. To this aim we should add to our typed semantics the rule

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

to advise the modeller that some unwanted behavior is happening in the system. In this way, the modeller can check if, starting from the initial term and using the given rules, we can reach a non-typable term, i.e. a term that breaks some biological property.
In biology, we do not always know the precise numbers of elements in the system, but we know their concentration, as percentage of the single elements in the whole system: for example, the corpuscles in blood are usually given as a percentage or as an absolute number per litre. Our type discipline cannot manage these cases, because it checks the exact numbers of elements in every compartment. As a possible future development, we plan to modify our type discipline to work on these cases, checking, in every compartment, not the exact numbers, but the ratio of elements with respect to the other elements.

Acknowledgement The author is grateful to the anonymous reviewers for their valuable comments. The final version of this paper is strongly improved due to their right criticisms and their suggestions. In particular the Section 3 of the present paper was completely rewritten, using multiset arithmetic and expanding the explanations of the Definitions 8-11. Moreover we added a more complete inference example in Section 5. A final and huge acknowledgement must go to Mariangiola Dezani and Paola Giannini, for proofreading with patience the drafts and highlighting conceptual and lexical mistakes.

References

[1] 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, 2001.
[2] B. Aman, M. Dezani-Ciancaglini, and A. Troina. Type Disciplines for Analysing Biologically Relevant Properties. In *Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'08)*, volume 227 of *ENTCS*, pages 97–111, 2009.
[3] R. Barbuti, A. Maggiolo-schettini, and P. Milazzo. Extending the calculus of looping sequences to model protein interaction at the domain level. In *Proceedings of International Symposium on Bioinformatics Research and Applications (ISBRA'07)*, volume 4463 of *LNBI*, pages 638–649, 2006.
[4] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. Bisimulation congruences in the calculus of looping sequences. In *Proceedings of International Colloquium on Theoretical Aspects of Computing (ICTAC'06)*, volume 4281 of *LNCS*, pages 93–107, 2006.
[5] 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.

[6] L. Bioglio, M. Dezani-Ciancaglini, P. Giannini, and A. Troina. Type directed semantics for the calculus of looping sequences. *International Journal of Software and Informatics*, 2010. to appear.

[7] L. Cardelli. Brane calculi - interactions of biological membranes. In *Computational Methods in Systems Biology*, pages 257–278. 2004.

[8] V. Danos and C. Laneve. Formal molecular biology. *Theoretical Computer Science*, 325:69–110, 2004.

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

[10] H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid Petri Net representation of gene regulatory networks. In *Proceedings of the Pacific Symposium on Biocomputing (PSB '00)*, pages 338–349, 2000. http://www.marmisicc.com/index.aspx?IDMenu=76&idMenuAPP=11.

[11] P. Milazzo. *Qualitative and Quantitative Formal Modeling of Biological Systems*. PhD thesis, University of Pisa, 2007.

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

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

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

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

[16] A. Syropoulos. Mathematics of multisets. In *Multiset Processing*, volume 2235 of *LNCS*, pages 347–358, 2001.

[17] J. B. Wells. The essence of principal typings. In *Proceedings of Intenational Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *LNCS*, pages 913–925, 2002.

Communicated by (The editor will be set by the publisher).
(The dates will be set by the publisher).