Typed Reductions of CLS*

Livio Bioglio

Dipartimento di Informatica, Università di Torino
bioglio.livio@educ.di.unito.it

Abstract. The calculus of looping sequences is a formalism for describing the evolution of biological systems by means of term rewriting rules. We enrich this calculus with a type discipline, derived from the requirement of certain elements and the repellency of others, and a type semantic, based on rules respecting different biological properties.

1 Introduction

Biologists usually describe biological systems by mathematical means, to reason on the behaviour of the systems and to perform simulations: when the complexity of the systems increase, these models become very difficult to manage. For this motivation, it is began to use Computer Science formalisms for the description of biological systems [13], that moreover permits the application of analysis methods that are practically unknown to biologists, such as static analysis and model checking.
The most notable formalisms applied to or created for biological systems are automata-based models [1,9], that permit the direct use of many verification tools such as model checkers, rewrite systems [8,11], that can be easily understood by biologists, and process calculi [13,14,12,7], that allows studying the behaviour of a system componentwise.
Milazzo et al. [4,5,10] developed a new formalism, called Calculus of Looping Sequences (CLS for short), for describing biological systems and their evolution. CLS mixes rewrite systems, because it is based on term rewriting, and process calculi, which uses some features, such as a commutative parallel composition operator, and some semantic means, such as bisimulations. This permits to combine the simplicity of notation of rewrite systems with the advantage of a form of compositionality.
In nature, there may be elements that always require the presence of other elements, such as pairs of oxygen atoms, or that always exclude the presence other ones, such as the blood type: it has inspired two extensions of CLS, proposed in [2] and improved in [3], in which every element type has 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. In this paper we borrow the type system in [3] to define a new typed semantic, that mixes up the semantics of [2] and [3].

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

2 Typed CLS

For a definition of CLS formalism we refer to [4]: here we only give the definition of patterns and terms, starting from a set $\mathcal{E}$ of elements.

Definition 1 (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 term variables, sequence variables and element variables, respectively. Terms are patterns without variables.

An example of CLS pattern is $(a \cdot \widetilde{x})^L \rfloor y \,|\, Z$: by applying the substitution $\widetilde{x} = b \cdot c$, $y = d$, $Z = e \,|\, f$ we obtain the CLS term $(a \cdot b \cdot c)^L \rfloor d \,|\, e \,|\, f$, that describes a biological system in which a membrane, formed by the elements a, b and c, containing the element d, is in juxtaposition with the elements e and f.

We classify every element in $\mathcal{E}$, representing molecule or other basic biological element, with a *basic type*, a type $\mathtt{t}$ which specifies the kind of the molecule: that fixed typing is reported in Γ. For each basic type $\mathtt{t} \in \Gamma$ 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 type $\mathtt{t}$ locally, that is in the same compartment, requires and forbids the presence of elements whose type is in $\mathtt{R_t}$ and in $\mathtt{E_t}$, respectively. *The types are pairs* $(\mathtt{P}, \mathtt{R})$, where $\mathtt{P}$ is the set of basic types of *present* elements, and $\mathtt{R}$ is the set of basic types of *required* elements: the set of *excluded* elements is implicitly given by $\mathtt{E_P} = \bigcup_{\mathtt{t} \in \mathtt{P}} \mathtt{E_t}$. Types are well formed, and pair of types are compatible, if their constraints on required and excluded elements are not contradictory; compatible types can be combined.

Definition 2 (Auxiliary definitions)
- *A type* $(\mathtt{P}, \mathtt{R})$ *is* well formed *if* $\mathtt{P} \cap \mathtt{E_P} = \mathtt{P} \cap \mathtt{R} = \mathtt{R} \cap \mathtt{E_P} = \emptyset$.
- *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* $(\mathtt{P}, \mathtt{R}) \sqcup (\mathtt{P'}, \mathtt{R'}) = (\mathtt{P} \cup \mathtt{P'}, (\mathtt{R} \cup \mathtt{R'}) \setminus (\mathtt{P} \cup \mathtt{P'}))$.

Basis are defined by:

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

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 patterns, and so terms and sequences, using the typing rules of Figure 1. All the rules are obvious except for the *loop*. In this rule we can put a pattern P inside a looping sequence SP only if all the types required from P are provided by SP: this is because if P gets inside a compartment (represented

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

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

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

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

Fig. 1. Typing rules for Present/Required Elements

by the looping sequence) it cannot interact any more with the environment.

We want to study only correct terms, whose type is well formed and whose requirements are completely satisfied. We start from correct terms, so to reach only correct terms we can require that a rewrite rule must not change the type of a term. To do that the left and the right hand of a rewrite rule must have the same type:

Definition 3 (Δ-safe rules) *A rewrite rule $P_1 \mapsto P_2$ is a Δ-safe rule if $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$ and $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$.*

Since a rule is applied to the hole of a context, another way to assure correctness preservation, without type preservation, is to check if the term obtained from a context by replacing the hole with a well typed term is correct:

Definition 4 (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 $\mathtt{P}'$.*

So, if we want to reach correct terms, it is enough to apply to the context a rule whose type of the right hand side is OK for that context:

Definition 5 (Δ-$(\mathtt{P}, \mathtt{R})$-rules) *A rule $P_1 \mapsto P_2$ is a Δ-$(\mathtt{P}, \mathtt{R})$-*rule *if $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$.*

An instantiation σ *agrees* with a basis Δ (notation $\sigma \in \Sigma_\Delta$) if $\rho : (\mathtt{P}, \mathtt{R}) \in \Delta$ implies $\vdash \sigma(\rho) : (\mathtt{P}, \mathtt{R})$.
We have found two ways to ensure correctness preservation: we can apply both Δ-safe rules inside arbitrary contexts, and Δ-$(\mathtt{P}, \mathtt{R})$-rules inside $(\mathtt{P}, \mathtt{R})$ OK contexts. The semantic for the type discipline therefore contains two rules, one for each way:

Definition 6 (Typed Semantics) *Given a finite set of rewrite rules $\mathcal{R}$, the* typed semantics *of CLS is the least relation closed with respect to $\equiv$ and satisfying the following rules:*

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

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

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

Theorem 1 *If* $\vdash T : (\mathtt{P}, \emptyset)$ *and* $T \Longrightarrow T'$*, then* $\vdash T : (\mathtt{P}', \emptyset)$ *for some* $\mathtt{P}'$*.*

2.1 Type Inference

In order to infer both which rules are Δ-safe rules and which ones are Δ-$(\mathtt{P},\mathtt{R})$ rules, we use the machinery of principal typing [15].

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 $\eta \in TV \cup SV$ there are two variables ϕ_η, ψ_η (called *p-type variable* and *r-type variable*) ranging over sets of basic types. Moreover we convene that Φ ranges over unions and differences of sets of basic types, e-type variables and p-type variables, and Ψ ranges over unions and differences of sets of basic types and r-type variables.
A *basis scheme* Θ is a mapping from atomic variables to their e-type variables, and from sequence and term variables to pairs of their p-type variables and r-type variables:

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

The rules for inferring principal typing use judgments 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 2 gives these inference rules. 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 2 (Soundness of Type Inference) *If* $\vdash P : \Theta; (\Phi, \Psi); \Xi$ *and* m *is a type mapping which satisfies* Ξ*, then* $\mathsf{m}(\Theta) \vdash P : (\mathsf{m}(\Phi), \mathsf{m}(\Psi))$*.*

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

A rule is safe if the principal types of the left and the right hand are equal, and if the variables in common have the same type. If there is a type mapping m that satisfies that constraints, a basis induced by m is surely safe for the rule: if also an instantiation agrees with this basis, then the application of the rule surely not change the type of the resulting term. With this idea, it is easy

$$\vdash \epsilon : \emptyset; (\emptyset, \emptyset); \emptyset \quad (\epsilon^*) \qquad \vdash x : \{x : (\varphi_x, \Psi))\} : (\varphi_x, \Psi); \{\Psi = \mathtt{R}_{\varphi_x}\} \quad (x^*)$$

$$\frac{a : \mathtt{t} \in \Gamma}{\vdash a : \emptyset; (\mathtt{t}, \mathtt{R_t}); \emptyset}\,(a^*) \quad \vdash \eta : \{\eta : (\phi_\eta, \psi_\eta)\}; (\phi_\eta, \psi_\eta); \emptyset \quad (\eta^*)$$

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

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

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

Fig. 2. Inference Rules for Principal Typing

to demonstrate that if the type mapping built from the instantiation derived from the term satisfies the constraints described above, then a rule can be safely applied to the term.

Theorem 4 (Applicability of Δ-safe Rules) *Let*

$$\vdash P_1 : \Theta; (\Phi; \Psi); \Xi \text{ and } \vdash P_2 : \Theta'; (\Phi'; \Psi'); \Xi'.$$

Then the rule $P_1 \mapsto P_2$ can be applied to the well typed term $C[P_1\sigma]$ 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 $\Xi \cup \Xi' \cup \{\Phi = \Phi'\} \cup \{\Psi = \Psi'\} \cup \{\tau = \tau' \mid \lambda : \tau \in \Theta \wedge \lambda : \tau' \in \Theta'\}$.

For deciding the OK relation it is only necessary to consider the part of the context influenced by the typing of the hole. Looking at the inference 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* that part:

Definition 7 *The* core of the context C *(notation* $\mathsf{core}(C)$*) is defined by:*

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

Since the core of a context is either the context itself or it does not influence the typing of the context, for checking if a type $(\mathtt{P}, \mathtt{R})$ is OK for a context C it is enough only to check if its core, assuming a typing $(\mathtt{P}, \mathtt{R})$ for the hole, is well typed, and if the core is the context itself also if its request set is empty; moreover there must be a term T such that the term obtained by replacing the hole with T in the context is correct, but, since we apply a rule only to a correct

term, that condition must not be checked.

With this idea, it is easy to demonstrate that a rule can be safely applied to a correct term if the core of the context derived from the term, assuming for its hole the typing derived for the left hand of the rule, satisfied the constraints described above.

Theorem 5 (Applicability of Δ-$(\mathtt{P},\mathtt{R})$ Rules) *Let*

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

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

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

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

According to the core definition, let $\mathsf{core}(C) \equiv (S_2)^L \rfloor ((S_1)^L \rfloor (\Box \,|\, T_1) \,|\, T_2)$, 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 we get the following six constraints for $\mathsf{core}(C)[X]$:

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

2.2 Analysis of the rules

There are some differences between the two rules used to define the typed semantic, that involve the use of rewrite rules and the inference mechanism.

First of all, the left and the right pattern of a Δ-safe rule must have the same type, to assure that the type of a term can not be modified by applying a Δ-safe rule. Logically it can be a right constraint, but since the type discipline checks only local properties, the type of a term, in particular the set of presents, is not derived from the whole term, but only from the terms outside the outermost looping sequence. Moreover, CLS is an high level model, so it can be useful to model creation and degradation of elements: it is not possible in Δ-safe rules. For example, there is no basis that can make safe the simple rule $a \,|\, b \mapsto c$.
On the contrary, there is no link between the types of the patterns in Δ-$(\mathtt{P},\mathtt{R})$-rules: this allows the use of more rules than the safe condition, with lower constraints. In particular, it is possible to construct rules of movement of elements through membranes, very useful to model biological systems: it is possible, for example, to apply the rules $(a{\cdot}b)^L \rfloor (c \,|\, X) \mapsto (a{\cdot}b{\cdot}c)^L \rfloor X$ and $(a{\cdot}b)^L \rfloor (c \,|\, X) \mapsto c \,|\, (a{\cdot}b)^L \rfloor X$ to a term.

Second, the set of constraints that must be satisfied to assure the safety of a rule is derived once, and checked for every instantiation where the rule is

applied. In this way, there is no need of run-time derivation, but is enough to check that the current instantiation satisfies the constraints derived once, in a previous phase. However, the number of constraints can be very high, for rules that involve a lot of variables.
In Δ-$(\mathtt{P},\mathtt{R})$-rules there is no way to decide a priori the whole set of constraints that must be satisfied to assure that the application of a rule to a correct term produce another correct term, but for every term where we want to apply the rule we must take the context and derive the set of constraints, valid only for that particular context, and difficultly reusable. However, the constraints derived from left hand of the rules can be derived once, and the number of constraint to derive for the context, thanks to the core idea, is limited by six.

Finally, it is possible to prove that every time the requirements for applying a Δ-safe rule are valid, the requirements for applying a Δ-$(\mathtt{P},\mathtt{R})$ rule are valid too.

Theorem 6 *If*

$$P_1 \mapsto P_2 \text{ is a } \Delta\text{-safe rule} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma_\Delta \quad C \in \mathcal{C} \quad C[P_1\sigma] : (\mathtt{P}', \emptyset)$$

then there is a type $(\mathtt{P},\mathtt{R})$ *such that* $P_1 \mapsto P_2$ *is a* Δ-$(\mathtt{P},\mathtt{R})$ *rule and* $(\mathtt{P},\mathtt{R})$ *is OK for* C.

This theorem proves that the terms derived by only Δ-safe rules are a subset of the terms derived by only Δ-$(\mathtt{P},\mathtt{R})$ rules: for this reason, during inference we can first check if the rule is Δ-safe, using constraints derived once, and only if it is not then we check if the rule is a Δ-$(\mathtt{P},\mathtt{R})$ rule, using run-time evaluation for the context. We can summarize this idea in a simple algorithm:

- in an initial phase, for every rule $P_1 \mapsto P_2 \in \mathcal{R}$, we infer $\vdash P_1 : \Theta; (\Phi, \Psi); \Xi$ and $\vdash P_2 : \Theta'; (\Phi', \Psi'); \Xi'$.
- during derivation, for every rule
  1. we check if the type mapping m, derived 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 and Ξ': if not, the rule is not applicable;
  2. we check if m satisfies $\{\Phi = \Phi'\} \cup \{\Psi = \Psi'\} \cup \{\tau = \tau' \mid \lambda : \tau \in \Theta \wedge \lambda : \tau' \in \Theta'\}$: if it is, the rule is a Δ-safe rule, and we can apply it, else we continue;
  3. we infer $\vdash core(C)[X] : \{X : (\phi_X, \psi_X)\}; (\Phi'', \Psi''); \Xi''$, and we check if m satisfies $\Xi'' \cup \{\Phi' = \phi_X, \Psi' = \psi_X\} \cup \{\Psi'' = \emptyset$ if ϕ_X or ψ_X occurs in $\Psi''\}$: if it is, the rule is a Δ-$(\mathtt{P},\mathtt{R})$ rule, and we can apply it, otherwise the correctness of the new term is not guaranteed.

Theorem 6 also implies that using the semantic in Definition 6 we derive the same terms we can derive using a semantic with only Δ-$(\mathtt{P},\mathtt{R})$ rules, but, using the algorithm above, Δ-safe rules permit to reduce run-time evaluation.

References

1. Alur, R., Belta, C., Ivancic, F., Kumar, V., Mintz, M., Pappas, G.J., Rubin, H. and Schug, J. (2001) Hybrid modelling and simulation of biomolecular networks. *Proc. of Hybrid Systems: Computation and Control*, LNCS 2034, Springer, 19-32.
2. Aman, B., Dezani-Ciancaglini, M., Troina, A. (2008) Type Disciplines for Analysing Biologically Relevant Properties. *Proc. of International Meeting on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'08)*, ENTCS 227, Elsevier, 97-111.
3. Dezani-Ciancaglini, M., Giannini, P. and Troina, A. (2009) A Type System for Required/Excluded Elements in CLS. *Proc. of Developments in Computational Models (DCM'09)*, EPTCS, to appear.
4. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P.and Troina, A. (2006) A calculus of looping sequences for modelling microbiological systems. *Fundamenta Informaticae*, **72**, 21-35.
5. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P. and Troina, A. (2006) Bisimulation congruences in the calculus of looping sequences. *Proc. of International Colloquium on Theoretical Aspects of Computing (ICTAC'06)*, LNCS 4281, Springer, 93-107.
6. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., and Troina, A. (2008) Bisimulations in calculi modelling membranes. *Formal Aspects of Computing*, **20**(4-5), 351-377.
7. Cardelli, L. (2005) Brane calculi. Interactions of biological membranes. *Proc. of Comput. Methods in Systems Biology (CMSB'04)*, LNCS 3082, Springer, 257-280.
8. Danos, V. and Laneve, C. (2004) Formal molecular biology. *Theoretical Computer Science*, **325**, 69-110.
9. Matsuno, H., Doi, A., Nagasaki, M. and Miyano, S. (2000) Hybrid Petri net representation of gene regulatory network. *Proc. of Pacific Symposium on Biocomputing*, World Scientific Press, 341-352.
10. Milazzo, P. (2007) *Qualitative and quantitative formal modelling of biological systems*. Ph.D. Thesis, University of Pisa.
11. Păun, G. (2002) *Membrane computing. An introduction*. Springer, 2002.
12. Regev, A., Panina, E. M., Silverman, W., Cardelli, L. and Shapiro, E. (2004) BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science*, **325**, 141-167.
13. Regev, A. and Shapiro, E. (2002) Cells as computation. *Nature*, **419**, 343.
14. Regev, A. and Shapiro, E. (2004) The π-calculus as an abstraction for biomolecular systems. *Modelling in Molecular Biology*, Natural Computing Series, Springer, 219-266.
15. Wells, J. (2002). The Essence of Principal Typings. *Proc. of 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, LNCS 2380, Springer, 913-925.