Enhancing reaction systems: a process algebraic approach

Linda Brodo[1], Roberto Bruni[2], and Moreno Falaschi[3]

[1] Dipartimento di Scienze economiche e aziendali, Università di Sassari, Italy
[2] Dipartimento di Informatica, Università di Pisa, Italy
[3] Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche
Università di Siena, Italy

Abstract. In the area of Natural Computing, reaction systems are a qualitative abstraction inspired by the functioning of living cells, suitable to model the main mechanisms of biochemical reactions. This model has already been applied and extended successfully to various areas of research. Reaction systems interact with the environment represented by the context, and pose problems of implementation, as it is a new computation model. In this paper we consider the `link`-calculus, which allows to model multiparty interaction in concurrent systems, and show that it allows to embed reaction systems, by representing the behaviour of each entity and preserving faithfully their features. We show the correctness and completeness of our embedding. We illustrate our framework by showing how to embed a *lac* operon regulatory network. Finally, our framework can contribute to increase the expressiveness of reaction systems, by exploiting the interaction among different reaction systems.

Keywords: process algebras, reaction systems, multi-party interaction

1 Introduction

Natural Computing is an emerging area of research which has two main aspects: human designed computing inspired by nature, and computation performed in nature. Reaction Systems (RSs) [10] are a rewriting formalism inspired by the way biochemical reactions take place in living cells. This theory has already shown to be relevant in several different fields, such as computer science [18], biology [2,16,1,3], molecular chemistry [20]. Reaction Systems formalise the mechanisms of biochemical systems, such as *facilitation* and *inhibition.* As a qualitative approximation of the real biochemical reactions, they consider if a necessary reagent is or not present, and likewise they consider if an inhibiting molecule is or not present. The possible reactants and inhibitors are called 'entities'. RSs model in a direct way the interaction of a living cell with the environment (called 'context'). However, two RSs are seen as independent models and do not interact.

In this paper, we present an encoding from RSs, to the open multiparty process algebra cCNA,[4] a variant of the `link`-calculus [6,7] without name mobility.

[4] After 'chained Core Network Algebra'.

This formalism allows several processes to synchronise and communicate altogether, at the same time, with a new communicating mechanism based on links and link chains. Our initial motivation for introducing this mechanism was to encode Mobile Ambients [13], getting a much stronger operational correspondence than any available in the literature, such as the one in [11]. This allowed us to easily encode calculi for biology equipped with membranes, as in [8]. Process calculi have been used successfully to model biological processes, see [5] for a recent survey. We illustrate our embedding by means of some simple basic examples, and then we consider a more complex example, by modeling a RS representing a regulatory network for *lac* operon, presented in [16]. We also show that our embedding preserves the main features of RSs, and prove its correctness and completeness. Our main contributions are as follows:

– the behaviour of the context, for each single entity, can be specified in a recursive way as an ordinary process;
– we can express the behaviour of entity mutation, in such a way that the mutated entity s' can take part to only a subset of rules requiring entity s;
– with a little coding effort, two RSs can communicate; i.e. a subset of those entities that the context can provide, are then provided by a second RS;
– as our translation results in a cCNA system, from each state only one transition can be generated, thus the cCNA computation is fully deterministic.

The main drawback of our proposal, is that the cCNA translation is verbose. Nevertheless it is clear that our translation can be automatised by means of a proper front-end in an implementation of the `link`-calculus.

As we have remarked, in our translation, Reaction Systems get the ability to interact between them in a synchronized manner. This interaction is not foreseen in the basic RS framework, as it can only happen with the context. By exploiting recursion, the kind of interactions which can be defined can be complex and expressive. Example 2 and more in general the discussion in Section 8 show that the interaction between RSs can help to model new scenarios.

Related work We are not aware of any inference rules for the RS; it seems not the case that a deterministic behavior, as the one of the RS, once the initial state is fixed, could be defined by inference rules that classically are applied to define non-deterministic transition systems. The reversible computation paradigm for RS extends the RS framework by allowing backward computations as well as forward computations; for this goal a set of inference rules for rewriting logic has been defined in [?] to exactly keep trace of those elements that dissolve in the next computation step.

Structure of the paper. Section 2 describes RSs and their semantics (interactive processes). Section 3 describes briefly the cCNA process algebra and its operational semantics. Section 4 defines the embedding of RSs in cCNA processes and shows some simple examples to illustrate it. Section 5 shows a more complex example taken from the literature on RSs and illustrating a *lac* operon. Section 8 presents some features and advantages of our embedding for the compositionality of RSs. Finally, Section 9 discusses future work, and concludes.

2 Reaction Systems

Natural Computing is concerned with human-designed computing inspired by nature as well as with computation taking place in nature. The theory of Reaction Systems [10] was born in the field of Natural Computing to model the behaviour of biochemical reactions taking place in living cells. Despite its initial aim, this formalism has shown to be quite useful not only for modeling biological phenomena, but also for the contributions which is giving to computer science [18], theory of computing, mathematics, biology [2,16,1,3], and molecular chemistry [20]. Here we briefly review the basic notions of RSs, see [10] for more details.

The mechanisms that are at the basis of biochemical reactions and thus regulate the functioning of a living cell, are *facilitation* and *inhibition*. These mechanisms are reflected in the basic definitions of Reaction Systems.

Definition 1 (Reaction). *A reaction is a triplet $a = (R, I, P)$, where R, I, P are finite, non empty sets and $R \cap I = \emptyset$. If S is a set such that $R, I, P \subseteq S$, then a is a reaction in S.*

The sets R, I, P are also written R_a, I_a, P_a and called the *reactant set* of a, the *inhibitor set* of a, and the *product set* of a, respectively. All reactants are needed for the reaction to take place. Any inhibitor blocks the reaction if it is present. Products are the outcome of the reaction. Also, $R_a \cup I_a$ is the set of the resources of a and $rac(S)$ denotes the set of all reactions in S. Because R and I are non empty, all products are produced from at least one reactant and every reaction can be inhibited in some way. Sometimes artificial inhibitors are used that are never produced by any reaction. For the sake of simplicity, in some examples, we will allow I to be empty.

Definition 2 (Reaction System). *A Reaction System (RS) is an ordered pair $\mathcal{A} = (S, A)$ such that S is a finite set, and $A \subseteq rac(S)$.*

The set S is called the *background set* of $\mathcal{A}$, its elements are called *entities*, they represent molecular substances (e.g., atoms, ions, molecules) that may be present in the states of a biochemical system. The set A is the set of *reactions* of $\mathcal{A}$. Since S is finite, so is A: we denote by $|A|$ the number of reactions in A.

Definition 3 (Reaction Result). *Let T be a finite set.*

1. *Let a be a reaction. Then a is enabled by T, denoted by $en_a(T)$, if $R_a \subseteq T$ and $I_a \cap T = \emptyset$. The result of a on T, denoted by $res_a(T)$, is defined by: $res_a(T) = P_a$, if $en_a(T)$, and $res_a(T) = \emptyset$ otherwise.*
2. *Let A be a finite set of reactions. The result of A on T, denoted by $res_A(T)$, is defined by: $res_A(T) = \bigcup_{a \in A} res_a(T)$.*

The theory of Reaction Systems is based on the following assumptions.

– **No permanency.** An entity of a set T vanishes unless it is sustained by a reaction. This reflects the fact that a living cell would die for lack of energy, without chemical reactions.

– **No counting.** The basic model of RSs is very abstract and qualitative, i.e. the quantity of entities that are present in a cell is not taken into account.
– **Threshold nature of resources.** From the previous item, we assume that either an entity is available and there is enough of it (i.e. there are no conflicts), or it is not available at all.

The dynamic behaviour of a RS is formalized in terms of *interactive processes*.

Definition 4 (Interactive Process). *Let $\mathcal{A} = (S, A)$ be a RS and let $n \geq 0$. An n-step* interactive process *in $\mathcal{A}$ is a pair $\pi = (\gamma, \delta)$ of finite sequences s.t. $\gamma = \{C_i\}_{i\in[0,n]}$ and $\delta = \{D_i\}_{i\in[0,n]}$ where $C_i, D_i \subseteq S$ for any $i \in [0, n]$, $D_0 = \emptyset$, and $D_i = res_{\mathcal{A}}(D_{i-1} \cup C_{i-1})$ for any $i \in [1, n]$.*

Living cells are seen as open systems that continuously react with the external environment, in discrete steps. The sequence γ is the *context sequence* of π and represents the influence of the environment on the Reaction System. The sequence δ is the *result sequence* of π and it is entirely determined by γ and A. The sequence $\tau = W_0, \dots, W_n$ with $W_i = C_i \cup D_i$, for any $i \in [0, n]$ is called a *state sequence*. Each state W_i in a state sequence is the union of two sets: the context C_i at step i and the result of the previous step.

For technical reasons, we extend the notion of an interactive process to deal with infinite sequences.

Definition 5 (extended interactive process). *Let $\mathcal{A} = (S, A)$ be a RS, and let $\pi = (\gamma, \delta)$ be an n-step interactive process, with $\gamma = \{C_i\}_{i\in[0,n]}$ and $\delta = \{D_i\}_{i\in[0,n]}$ Then, let $\pi' = (\gamma', \delta')$ be the extended interactive process of $\pi = (\gamma, \delta)$, defined as $\gamma' = \{C'_i\}_{i\in\mathbb{N}}$, $\delta' = \{D'_i\}_{i\in\mathbb{N}}$, where $C'_j = C_j$ for $j \in [0, n]$ and $C'_j = \emptyset$ for $j > n$, $D'_0 = D_0$ and $D'_j = res_A(D'_{j-1} \cup C'_{j-1})$ for $j \geq 1$.*

3 Chained CNA (cCNA)

In this section we introduce the syntax and operational semantics of a variant of the `link`-calculus [6], the cCNA (chained CNA) where the prefixes are link chains.

Link Chains. Let $\mathcal{C}$ be the set of channels, ranged over by $a, b, \ldots$, and let $\mathcal{A} = \mathcal{C} \cup \{\tau\} \cup \{\square\}$ be the set of actions, ranged over by $\alpha, \beta, \ldots$, where the symbol τ denotes a *silent* action, while the symbol $\square$ denotes a *virtual* (non-specified) action. A *link* is a pair $\ell = {}^{\alpha}\backslash_{\beta}$; it is *solid* if $\alpha, \beta \neq \square$; the link ${}^{\square}\backslash_{\square}$ is called *virtual*. A link is *valid* if it is solid or virtual. We let $\mathcal{L}$ be the set of valid links. A *link chain* is a finite sequence $v = \ell_1 \ldots \ell_n$ of (valid) links $\ell_i = {}^{\alpha_i}\backslash_{\beta_i}$ such that:

1. for any $i \in [1, n-1]$, $\begin{cases} \beta_i, \alpha_{i+1} \in \mathcal{C} & \text{implies } \beta_i = \alpha_{i+1} \\ \beta_i = \tau & \text{iff } \alpha_{i+1} = \tau \end{cases}$
2. $\exists i \in [1, n].\ \ell_i \neq {}^{\square}\backslash_{\square}$.

Virtual links represent missing elements of a chain. The equivalence $\bowtie$ models expansion/contraction of virtual links to adjust the length of a link chain.

Definition 6 (Equivalence $\bowtie$). *We let $\bowtie$ be the least equivalence relation over link chains closed under the axioms (whenever both sides are well defined):*

$$\begin{array}{rcl@{\qquad}rcl}
v{}^{\square}\backslash_{\square} & \bowtie & v & v_1{}^{\square}\backslash^{\square}_{\square}\backslash_{\square}v_2 & \bowtie & v_1{}^{\square}\backslash_{\square}v_2 \\
{}^{\square}\backslash_{\square}v & \bowtie & v & v_1{}^{\alpha}\backslash^{a}_{a}\backslash_{\beta}v_2 & \bowtie & v_1{}^{\alpha}\backslash^{\square}_{a}\backslash^{a}_{\square}\backslash_{\beta}v_2
\end{array}$$

Two link chains of equal length can be merged whenever each position occupied by a solid link in one chain is occupied by a virtual link in the other chain and solid links in adjacent positions match. Positions occupied by virtual links in both chains remain virtual. Merging is denoted by $v_1 \bullet v_2$. For example, given $v_1 = {}^{a}\backslash^{\square}_{b}\backslash^{\square}_{\square}\backslash_{\square}$ and $v_2 = {}^{\square}\backslash^{b}_{\square}\backslash^{\square}_{c}\backslash_{\square}$ we have $v_1 \bullet v_2 = {}^{a}\backslash^{b}_{b}\backslash^{\square}_{c}\backslash_{\square}$.

Some names in a link chain can be restricted as non observable and transformed into silent actions τ. This is possible only if they are matched by some adjacent link. Restriction is denoted by $(\nu\, a)v$. For example, given $v = {}^{a}\backslash^{b}_{b}\backslash^{\square}_{c}\backslash_{\square}$ we have $(\nu\, b)v = {}^{a}\backslash^{\tau}_{\tau}\backslash^{\square}_{c}\backslash_{\square}$.

Syntax. The cCNA processes, denoted as $\mathcal{P}$, are generated by the following grammar:

$$P, Q ::= \textstyle\sum_{i \in I} v_i.P_i \;\mid\; P|Q \;\mid\; (\nu\, a)P \;\mid\; P[\phi] \;\mid\; A$$

where v_i is a link chain, ϕ is a channel renaming function, and A is a process identifier for which we assume a definition $A \triangleq P$ is available in a given set Δ of (possibly recursive) process definitions. We let $\mathbf{0}$, the inactive process, denote the empty summation.

The syntax of cCNA extends that of CNA [7] by allowing to use link chains as prefixes instead of links. For the rest it features nondeterministic choice, parallel composition, restriction, relabelling and possibly recursive definitions. Here we do not consider name mobility, which is present instead in the `link`-calculus.

Semantics. The operational semantics of cCNA is defined in the SOS style by the inference rules in Fig.1. The rules are reminiscent of those for Milner's CCS and they essentially coincide with those of CNA in [7]. The only difference is due to the presence of prefixes that are link chains. Briefly: rule (*Sum*) selects one alternative and puts as label a possible contraction/expansion of the link chain in the selected prefix; rule (*Ide*) selects one transition of the defining process for a constant; rule (*Res*) restricts some names in the label (it cannot be applied when $(\nu\, a)v$ is not defined); rules (*Lpar*) and (*Rpar*) account for interleaving in parallel composition; rule (*Com*) synchronises interactions (it cannot be applied when $v \bullet v'$ is not defined).

Analogously to CNA, the operational semantics of cCNA satisfies the so called Accordion Lemma: whenever $P \xrightarrow{v} P'$ and $v' \bowtie v$ then $P \xrightarrow{v'} P'$.

3.1 Notation for link chains

Hereafter we make use of some new notations for link chains that will facilitate the presentation of our translation.

$$\frac{\upsilon \bowtie \upsilon_j \quad j \in I}{\sum_{i\in I} \upsilon_i.P_i \xrightarrow{\upsilon} P_j}\ (Sum) \qquad \frac{P \xrightarrow{\upsilon} P' \qquad (A \triangleq P) \in \Delta}{A \xrightarrow{\upsilon} P'}\ (Ide)$$

$$\frac{P \xrightarrow{\upsilon} P'}{(\nu\, a)P \xrightarrow{(\nu\, a)\upsilon} (\nu\, a)P'}\ (Res) \quad \frac{P \xrightarrow{\upsilon} P'}{P|Q \xrightarrow{\upsilon} P'|Q}\ (Lpar) \quad \frac{P \xrightarrow{\upsilon'} P' \qquad Q \xrightarrow{\upsilon} Q'}{P|Q \xrightarrow{\upsilon\bullet\upsilon'} P'|Q'}\ (Com)$$

Fig. 1: SOS semantics of the cCNA (rules (*Rel*) and (*Rpar*) omitted).

Definition 7 (Replication). *Let υ be a link chain. Its n times replication υ^n is defined recursively by letting $\upsilon^0 = \epsilon$ (i.e. the empty chain) and $\upsilon^n = \upsilon^{n-1}\upsilon$, with the hypothesis that all the links in the resulting link chains match.*

For example, the expression $({}^{a}\backslash^{\square}_{b}\backslash_{\square})^3$ denotes the chain ${}^{a}\backslash^{\square}_{b}\backslash^{a}_{\square}\backslash^{\square}_{b}\backslash^{a}_{\square}\backslash^{\square}_{b}\backslash_{\square}$. We introduce the *half link* that will be used in conjunction with the *open block of chain* to form regular link chains.

Definition 8 (Half links). *Let a be a channel name, we define the* half left link*: ${}^{a}\backslash$, and the* half right link*: $\backslash_{a}$.*

Definition 9 (Open block). *Let R be a set of names. We define an* open block *as $\big(\mathop{\backslash\!\!\backslash}_{a\,\in\,R} {}^{\square}_{a_i}\backslash^{a_o}_{\square}\big)$, where a_i and a_o are annotated version of the name a, as*

set	*block of chain*	*result*
$R = \emptyset$	$\big(\mathop{\backslash\!\!\backslash}_{a\,\in\,R} {}^{\square}_{a_i}\backslash^{a_o}_{\square}\big)$	ϵ
$R = \{b\}$	$\big(\mathop{\backslash\!\!\backslash}_{a\,\in\,R} {}^{\square}_{a_i}\backslash^{a_o}_{\square}\big)$	${}^{\square}_{b_i}\backslash^{b_o}_{\square}$
$R = \{b\} \cup R'$	$\big(\mathop{\backslash\!\!\backslash}_{a\,\in\,R} {}^{\square}_{a_i}\backslash^{a_o}_{\square}\big)$	${}^{\square}_{b_i}\backslash^{b_o}_{\square}\backslash\ \big(\mathop{\backslash\!\!\backslash}_{a\,\in\,R'} {}^{\square}_{a_i}\backslash^{a_o}_{\square}\big)$

We then combine half links and open blocks to form regular link chains. For example, for $R = \{a, b\}$ the expression $\big(\mathop{\backslash\!\!\backslash}_{c\,\in\,R} {}^{\square}_{c_i}\backslash^{c_o}_{\square}\big)$ denotes the block of chains ${}^{\square}_{a_i}\backslash^{a_o}_{\square}\backslash^{\square}_{b_i}\backslash^{b_o}_{\square}$; and the expression ${}^{r_1}\backslash\ \big(\mathop{\backslash\!\!\backslash}_{c\,\in\,R} {}^{\square}_{c_i}\backslash^{c_o}_{\square}\big)\backslash_{r_2}$ denotes the chain ${}^{r_1}\backslash^{\square}_{a_i}\backslash^{a_o}_{\square}\backslash^{\square}_{b_i}\backslash^{b_o}_{\square}\backslash_{r_2}$.

4 From Reaction Systems to cCNA

Here we present a translation from Reaction Systems to cCNA. The idea is to define separated processes for representing the behaviour of each entity, each reaction, and for the provisioning of each entity by the context.

Processes for entities. Given an entity $s \in S$, we exploit four different names for the interactions over s: names s_i, s_o are used to test the presence of s in the system; names $\hat{s}_i$, $\hat{s}_o$ are used to test the provisioning of s from the context; names $\tilde{s}_i$, $\tilde{s}_o$ are used to test the production of s by some reaction; names $\overline{s}_i$, $\overline{s}_o$ are used to test the absence of s from the context; and names $\underline{s}_i$, $\underline{s}_o$ are used to test the absence of s in the system. We let P_s be the process implementing the presence of s in the system, and $\overline{P_s}$ be the one for its absence. They can be seen as instances of the same template, which is given below.

$$P_s \triangleq P(s, \tilde{s}, \hat{s}, \underline{s}) \qquad\qquad \overline{P_s} \triangleq P(\overline{s}, \tilde{s}, \hat{s}, \underline{s})$$

$$\begin{aligned}
P(s, \tilde{s}, \hat{s}, \underline{s}) \triangleq {} & \textstyle\sum_{h,k\geq 0} ({}^{s_i}\backslash^{\Box}_{s_o}\backslash_{\Box})^h \; {}^{\hat{s}_i}\backslash^{\Box}_{\hat{s}_o}\backslash_{\Box} ({}^{\tilde{s}_i}\backslash^{\Box}_{\tilde{s}_o}\backslash_{\Box})^k . P_s \\
& + \\
& \textstyle\sum_{h\geq 0,k\geq 1} ({}^{s_i}\backslash^{\Box}_{s_o}\backslash_{\Box})^h \; {}^{\underline{s}_i}\backslash^{\Box}_{\underline{s}_o}\backslash_{\Box} ({}^{\tilde{s}_i}\backslash^{\Box}_{\tilde{s}_o}\backslash_{\Box})^k . P_s \\
& + \\
& \textstyle\sum_{h\geq 0} ({}^{s_i}\backslash^{\Box}_{s_o}\backslash_{\Box})^h \; {}^{\underline{s}_i}\backslash_{\underline{s}_o} . \overline{P_s}
\end{aligned}$$

The first line of $P(s, \tilde{s}, \hat{s}, \underline{s})$ accounts for the case where s is tested for presence by h reactions and produced by k reactions, while being provided by the context (${}^{\hat{s}_i}\backslash_{\hat{s}_o}$). Thus, s will be present at the next step (the continuation is P_s). Here h and k are not known a priori and therefore any combination is possible. By knowing the number of reactions that test s, we can bound the maximum values of h and k. The second line accounts for the analogous case where s is not provided by the context (${}^{\underline{s}_i}\backslash_{\underline{s}_o}$). The condition $k \geq 1$ guarantees that s will remain present (the continuation is P_s). The third line accounts for the case where s is tested for presence, but it is neither produced nor provided by the context. Therefore, in the next step s will be absent in the system (the continuation is $\overline{P_s}$). Note that in the case of $\overline{P_s}$ the test for presence of s in the system is just replaced by the test for its absence.

Processes for reactions. We assume that each reaction a is assigned a progressive number j. The process for reaction $aj = (R_j, I_j, P_j)$ must assert either the possibility to apply the reaction or its impossibility. The first case happens when all its reactants are present (the link ${}^{s_i}\backslash_{s_o}$ is requested for any $s \in R_j$) and all its inhibitors are absent (the link ${}^{\overline{e}_i}\backslash_{\overline{e}_o}$ is requested for any $e \in I_j$), then the product set is released (the link ${}^{\tilde{c}_i}\backslash_{\tilde{c}_o}$ is requested for any $c \in P_j$). The next case can happen for two reasons: one of the reactants is absent (the link ${}^{\overline{s}_i}\backslash_{\overline{s}_o}$ is requested for some $s \in R_j$) or one of the inhibitors is present (the link ${}^{e_i}\backslash_{e_o}$ is requested for some $e \in I_j$). The process is recursive so that reactions can be applied at any step.

$$\begin{aligned}
P_{aj} \triangleq {} & {}^{r_j}\backslash \Big(\mathop{\Big\|}_{s\,\in\,R_j} {}^{\Box}_{s_i}\backslash^{s_o}_{\Box}\Big)\backslash \Big(\mathop{\Big\|}_{e\,\in\,I_j} {}^{\Box}_{\overline{e}_i}\backslash^{\overline{e}_o}_{\Box}\Big)\backslash^{\Box}_{r_{j+1}}\backslash^{p_j}_{\Box}\backslash \Big(\mathop{\Big\|}_{c\,\in\,P_j} {}^{\Box}_{\tilde{c}_i}\backslash^{\tilde{c}_o}_{\Box}\Big)\backslash_{p_{j+1}} . P_{aj} && \{aj \text{ is applicable}\} \\
& + \\
& \textstyle\sum_{s\in R_j} {}^{r_j}\backslash^{\Box}_{\overline{s}_i}\backslash^{\overline{s}_o}_{\Box}\backslash^{\Box}_{r_{j+1}}\backslash^{p_j}_{\Box}\backslash_{p_{j+1}} . P_{aj} && \{aj \text{ is not applicable}\} \\
& + \\
& \textstyle\sum_{e\in I_j} {}^{r_j}\backslash^{\Box}_{e_i}\backslash^{e_o}_{\Box}\backslash^{\Box}_{r_{i+1}}\backslash^{p_j}_{\Box}\backslash_{p_{j+1}} . P_{aj} && \{aj \text{ is not applicable}\}
\end{aligned}$$

We exploit names r_j, p_j to join the chains provided by the application of all the reactions. Channels r_j and r_{j+1} enclose the enabling/disabling condition of reaction aj. Channels p_i and p_{j+1} enclose the links related to the entities produced by aj. We will see that all the link chain labels of transitions follow the same schema: first we find all the reactions limited to the reactants and inhibitors (chained using r_j channels), then all the supplies by the contexts (chained using cxt_j channels, to be introduced next), and finally the products for all the reactions (chained using p_j channels). In the following there is an example explaining this schema.

Processes for contexts. For each entity $s \in S$, we introduce another process Cxt_s, participating in each transition and saying if, the entity s is provided by the context or not. As done for the reactions, we assume that entities are enumerated and use the names cxt_j to concatenate the chains formed by the application of all the contexts. For each entity s with number j, at step $n > 0$ there are two possible behaviours:

$$Cxt^n \triangleq \sum_j^{cxt} \backslash \Big(\bigvee\!\!\!\bigvee_{s \,\in\, I_{j_n}} \frac{\square}{s_i} \backslash \frac{s_o}{\square} \Big) \backslash \Big(\bigvee\!\!\!\bigvee_{e \,\in\, O_{j_n}} \frac{\square}{\overline{e}_i} \backslash \frac{\overline{e}_o}{\square} \Big) \backslash_{p_1} . Cxt^{n+1}$$

We only consider Cxt^n with $n > 0$, as the entities that are present at step zero are considered to be present in the initial system (process P_s instead of $\overline{P_s}$).

Definition 10 (Translation). *Let $\mathcal{A} = (S, A)$ be a RS, and let $\pi = (\gamma, \delta)$ be an extended interactive process in $\mathcal{A}$, with $\gamma = \{C_i\}_{i \in \mathbb{N}}$. We define its cCNA translation $[\![\mathcal{A}, \gamma]\!]$ as follows:*

$$[\![\mathcal{A}, \gamma]\!] = (\nu\, reacts, ctxs, ents, prods)(\varPi_{s \in C_0} P_s | \varPi_{s \notin C_0} \overline{P_s} | \varPi_{a \in A} P_a | \varPi_{s \in S} Cxt_s),$$

with reacts be the set of reaction names r_j, cxts the set of context names cxt_j, ents the set of decorated entity names $\{s_i, s_o, \hat{s}_i, \hat{s}_o, \tilde{s}_i, \tilde{s}_o, \overline{s}_i, \overline{s}_o, \underline{s}_i, \underline{s}_o | s \in S\}$, and prods be the set of names p_j associated to each reaction. In the following, we set $names = reacts \cup ctxs \cup ents \cup prods$. For notational convenience, we fix that $r_1 = \tau$, $r_{u+1} = cxt_1$ for u the number of reacts, and $cxt_{w+1} = p_1$ $p_{u+1} = \tau$ for w the number of entities.

It is important to observe that, for each transition, our cCNA encoding requires all the processes P_a, with $a \in A$, and Cxt_s and P_s, with $s \in S$, be interacting in that transition. This is due to the fact that all the channels r_j, p_j, cxt_h, and s_{hi}, and s_{ho} are restricted. Each reaction defines a pattern to be satisfied, i.e. each reaction inserts as many virtual links as the number of reactants, inhibitors, and products, as required by the corresponding reaction.

Lemma 1. *Let $\mathcal{A} = (S, A)$ be a RS and let $\pi = (\gamma, \delta)$ be an extended interactive process in $\mathcal{A}$. Let $P = [\![\mathcal{A}, \gamma]\!]$ its cCNA translation. If exists P' such that $t = (P \xrightarrow{(\nu\, names)\upsilon} P')$ is a transition of P, then*

1. *for each reaction $a_j \in A$, the corresponding channels r_j and p_j appear in υ; for each entity $s_h \in S$ (where h is the identifying number of s), the corresponding channel s_h (suitably decorated), and the corresponding channel cxt_h appear in υ;*
2. *for each reaction $a \in A$ and each entity $s \in S$, each virtual link offered by processes P_a and Cxt_s is overlapped by exactly one solid link offered by processes representing entities.*

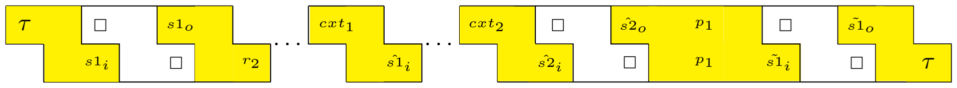

Fig. 2: The link chain structure arising from reactions and context processes.

Example 1. Let $\mathcal{A}$ be a RS whose specification contains two entities, $s1$ and $s2$, and, among the others, the reaction $a = (s1, \ , s1)$ that guarantees the presence of the $s1$ in the system. Then, we assume an extended interactive process $\pi = (\gamma, \delta)$ where the context γ provides $s1$ and $s2$. Our translation includes the processes:

$$
\begin{array}{llll}
P_a & \triangleq {}^{\tau}\backslash^{\square}_{s1_i}\backslash^{s1_o}_{\square}\backslash^{\square}_{r_2}\backslash^{p_1}_{\square}\backslash^{\square}_{\tilde{s1}_i}\backslash^{\tilde{s1}_o}_{\square}\backslash_{p_2}.P_a + \dots; & & \\
P_{s1} & \triangleq {}^{s1_i}\backslash^{\square}_{s1_o}\backslash^{\hat{s1}_i}_{\square}\backslash^{\square}_{\hat{s1}_o}\backslash^{\tilde{s1}_i}_{\square}\backslash_{\tilde{s1}_o}.P_{s1} + \dots; & P_{s2} & \triangleq {}^{\hat{s2}_i}\backslash_{\hat{s2}_o}.P_{s2} + \dots; \\
Cxt_{s1} & \triangleq {}^{cxt_1}\backslash^{\square}_{\hat{s1}_i}\backslash^{\hat{s1}_o}_{\square}\backslash_{cxt_2}.Cxt_{s1} & Cxt_{s2} & \triangleq {}^{cxt_2}\backslash^{\square}_{\hat{s2}_i}\backslash^{\hat{s2}_o}_{\square}\backslash_{p_1}.Cxt_{s2}
\end{array}
$$

Now, we assume that $s1$ is in the initial state of $\mathcal{A}$, and in Figure 2 we show the structure of a link chain label related to the execution of a transition of the cCNA system: $(\nu\, names)(P_{s1}|P_{s2}|P_a|\dots|Cxt_{s1}|Cxt_{s2})$. The yellow blocks are referred to the processes encoding the reactions (P_a, in our case) and the contexts (Cxt_{s1} and Cxt_{s2}). As the figure puts in evidence, these two kinds of processes determine the structure of the link chain, from end to end, i.e. from the left τ to the right one. We could say that these processes form the *backbone* of the interaction. In contrast, the processes encoding the entities (P_{s1}, and P_{s2}, in our case) provides the solid links to overlap the virtual links of the backbone.

Example 1 outlines two different roles of the processes defining the translation of an interactive process: those processes encoding the reactions and the context provide the backbone of each transition, whereas the processes encoding the entities provide the resources needed for the communication to take place.

With the next proposition, we analyse the structure of a cCNA process encoding of a reactive process after one step transition. In the following four statements, for brevity, we let $\mathcal{A} = (S, A)$ be a RS, and let $\pi = (\gamma, \delta)$ be an extended interactive process in A, with $\gamma = \{C_i\}_{i\in\mathbb{N}}$ and $\delta = \{D_i\}_{i\in\mathbb{N}}$. Moreover, we denote by π^j the shift of π starting at the j-th state sequence; formally we let $\pi^j = (\gamma^j, \delta^j)$ with $\gamma^j = \{C'_i\}_{i\in\mathbb{N}}$, $\delta^j = \{D'_i\}_{i\in\mathbb{N}}$ with $C'_0 = C_j \cup D_j$, and $C'_i = C_{i+j}$, $D'_i = D_{i+j}$ for any $i \geq 1$.

Proposition 1 (Correctness 1). *Let $P = [\![\mathcal{A}, \gamma]\!]$ with*
$P = (\nu\, names)(\Pi_{a\in A}P_a|\Pi_{s\in S}Cxt_s|\Pi_{s\in C_0}P_s|\Pi_{s\notin C_0}\overline{P}_s)$.
If there exists P' such that $P \xrightarrow{v} P'$, it holds that:
$v = {}^{\tau}\backslash_{\tau}\dots{}^{\tau}\backslash_{\tau}$ *and*
$P' = (\nu\, names)(\Pi_{a\in A}P_a|\Pi_{s\in S}Cxt_s|\Pi_{s\in C_1\cup D_1}P_s|\Pi_{s\notin C_1\cup D_1}\overline{P}_s)$.
Moreover, given $\pi^1 = (\gamma^1, \delta^1)$, we have $P' = [\![\mathcal{A}, \gamma^1]\!]$.

Now, we extend the previous result to a series of transitions.

Corollary 1 (Correctness 2). *Let $P = [\![\mathcal{A}, \gamma]\!]$ and $j \geq 1$. If there exists P'' such that $P \xrightarrow{{}^{\tau}\backslash_{\tau}\cdots{}^{\tau}\backslash_{\tau}}{}^{j} P''$, then letting $\pi^j = (\gamma^j, \delta^j)$ we have $P'' = [\![\mathcal{A}, \gamma^j]\!]$.*

With the following propositions, we prove that, given a RS $\mathcal{A} = (S, A)$ and an extended interactive process $\pi = (\gamma, \delta)$, then the cCNA process $[\![\mathcal{A}, \gamma]\!]$ can simulate all the evolutions of π.

Proposition 2 (Completeness 1). *Let $P = [\![\mathcal{A}, \gamma]\!]$ and $\pi^1 = (\gamma^1, \delta^1)$. Then, $P \xrightarrow{{}^{\tau}\backslash_{\tau}\cdots{}^{\tau}\backslash_{\tau}} P' = [\![\mathcal{A}, \gamma^1]\!]$.*

Now, we extend the previous result to a series of transitions.

Corollary 2 (Completeness 2). *Let $P = [\![\mathcal{A}, \gamma]\!]$ and $\pi^j = (\gamma^j, \delta^j)$. Then, $P \xrightarrow{{}^{\tau}\backslash_{\tau}\cdots{}^{\tau}\backslash_{\tau}}{}^{j} P'' = [\![\mathcal{A}, \gamma^j]\!]$.*

5 Example: *lac* operon

In this section we present the encoding of a RS example taken from [16].

5.1 The *lac* operon

An operon is a cluster of genes under the control of a single promoter. The *lac* operon is involved in the metabolism of lactose in *Escherichia coli* cells; it is composed by three adjacent structural genes (plus some regulatory components): *lacZ*, *lacY* and *lacA* encoding for two enzymes Z and A, and a transporter Y, involved in the digestion of the *lactose*. The main regulations are:

- the gene *lacI* encodes for a repressor protein I;
- the DNA sequence, called *promoter*, is recognised by a RNA polymerase to iniziate the transcription of the genes *lacZ*, *lacY* and *lacA*;
- a DNA segment, called the *operator* (OP), obstructs the RNA polymerase functionality when the repressor protein I is bound to it forming I-OP;
- a short DNA sequence, called the *CAP-binding site*, when it is bound to the complex composed by the protein CAP and the signal molecule $cAMP$, acts as a promoter for the interaction between the RNA polymerase and the promoter.

The functionality of the *lac* operon depends on the integration of two control mechanisms, one mediated by *lactose*, and the other one mediated by *glucose*.

In the first control mechanism, an effect of the absence of the *lactose* is that I is able to bind the operator sequence preventing the *lac* operon expression. If lactose is available, I is unable to bind the operator sequence, and the *lac* operon can be potentially expressed.

In the second control mechanism, when glucose is absent, the molecule $cAMP$ and the protein CAP increase the *lac* operon expression, thanks to the fact that the binding between the molecular complex $cAMP$-CAP and the *CAP-binding site* increases. In summary, the condition promoting the operon gene expression is when the *lactose* is present and the *glucose* is absent.

In the following we report the description of the *lac* operon mechanism in the reaction system formalism and then show its encoding in cCNA.

5.2 The RS formalization

The reaction system for the *lac* operon is defined as $A_{lac} = (S, A)$, where the set S represents the main biochemical components involved in this genetic system, while the reaction set A contains the biochemical reactions involved in the regulation of the *lac* operon expression. Formally, the *lac* operon reaction system is defined as follows: S is the set $\{lac, Z, Y, A, lacI, I, I\text{-}OP, cya, cAMP, crp, CAP, cAMP\text{-}CAP, lactose, glucose\}$, and A consists of the following 10 reactions:

$$
\begin{array}{ll}
a_1 = (\{lac\}, \{...\}, \{lac\}), & a_6 \;= (\{cya\}, \{...\}, \{cAMP\}), \\
a_2 = (\{lacI\}, \{...\}, \{lacI\}), & a_7 \;= (\{crp\}, \{...\}, \{crp\}), \\
a_3 = (\{lacI\}, \{...\}, \{I\}), & a_8 \;= (\{crp\}, \{...\}, \{CAP\}), \\
a_4 = (\{I\}, \{lactose\}, \{I\text{-}OP\}), & a_9 \;= (\{cAMP, CAP\}, \{glucose\}, \{cAMP\text{-}CAP\}), \\
a_5 = (\{cya\}, \{...\}, \{cya\}), & a_{10} = (\{lac, cAMP\text{-}CAP\}, \{I\text{-}OP\}, \{Z, Y, A\}).
\end{array}
$$

The default context (DC) is composed by those entities that are always present in the system $DC = \{lac, lacI, I, cya, cAMP, crp, CAP\}$, whereas the *lactose* and the *glucose* are given non-deterministically by the context.

5.3 The RS encoding

For the sake of readability, the encoding we propose exploits the specific features of the example in hand to perform some simplifications:

- for the entities in the default context, $s \in DC$, as they are persistent, we do not provide the $\overline{P_s}$ processes and the Cxt_s processes;
- for the reactions requiring the presence of entities $s \in DC$, we do not provide the reaction alternative behaviour for when s is absent;
- the Cxt_s processes are specified only for those entitiess that are really provided by the context.

Moreover, we do not model the *dummy* entity that is specified by dots $(\dots)$ by the RS reactions in Section 5.2. Finally, we exclude the *duplication reactions* (a_1, a_2, a_5, a_7), and renumber the remaining reactions :

old	new	reactions
a_3	a_1	$= (\{lacI\}, \{...\}, \{I\})$,
a_4	a_2	$= (\{I\}, \{lactose\}, \{I\text{-}OP\})$,
a_6	a_3	$= (\{cya\}, \{...\}, \{cAMP\})$,
a_8	a_4	$= (\{crp\}, \{...\}, \{CAP\})$,
a_9	a_5	$= (\{cAMP, CAP\}, \{glucose\}, \{cAMP\text{-}CAP\})$,
a_{10}	a_6	$= (\{lac, cAMP\text{-}CAP\}, \{I\text{-}OP\}, \{Z, Y, A\})$.

Duplication reactions. As explained previously, their encoding is omitted.

Expression reactions. First we define the parametric process

$$P_i(s1,s2) \triangleq {}^{r_i}\backslash^{\Box}_{s1_i}\backslash^{s1_o}_{\Box}\backslash^{\Box}_{r_{i+1}}\backslash^{p_i}_{\Box}\backslash^{\Box}_{\tilde{s2}_i}\backslash^{\tilde{s2}_o}_{\Box}\backslash_{p_{i+1}}.P_i(s1,s2)$$

Then, we let $P_{a1} \triangleq P_1(lacI, I)$, $P_{a3} \triangleq P_3(cya, cAMP)$, and $P_{a4} \triangleq P_4(crp, CAP)$.

Regulation reactions.

$$\begin{aligned}
P_{a2} \triangleq\ & {}^{r_2}\backslash^{\Box}_{I_i}\backslash^{I_o}_{\Box}\backslash^{\Box}_{\overline{lactose_i}}\backslash^{\overline{lactose}_o}_{\Box}\backslash^{\Box}_{r_3}\backslash^{p_2}_{\Box}\backslash^{\Box}_{\widetilde{I\text{-}OP}_i}\backslash^{\widetilde{I\text{-}OP}_o}_{\Box}\backslash_{p_3}.P_{a2}\\
& +\\
& {}^{r_2}\backslash^{\Box}_{lactose_i}\backslash^{lactose_o}_{\Box}\backslash^{\Box}_{r_3}\backslash^{p_2}_{\Box}\backslash_{p_3}.P_{a2} \ + \ {}^{r_2}\backslash^{\Box}_{\overline{I}_i}\backslash^{\overline{I}_o}_{\Box}\backslash^{\Box}_{r_3}\backslash^{p_2}_{\Box}\backslash_{p_3}.P_{a2}\\
P_{a5} \triangleq\ & {}^{r_5}\backslash^{\Box}_{cAMP_i}\backslash^{cAMP_o}_{\Box}\backslash^{\Box}_{CAP_i}\backslash^{CAP_o}_{\Box}\backslash^{\Box}_{\overline{glucose_i}}\backslash^{\overline{glucose}_o}_{\Box}\backslash^{\Box}_{r_6}\backslash^{p_5}_{\Box}\backslash^{\Box}_{\widetilde{cAMP\text{-}CAP}_i}\backslash^{\widetilde{cAMP\text{-}CAP}_o}_{\Box}\backslash_{p_6}.P_{a5}\\
& +\\
& {}^{r_5}\backslash^{\Box}_{glucose_i}\backslash^{glucose_o}_{\Box}\backslash^{\Box}_{r_6}\backslash^{p_5}_{\Box}\backslash_{p_6}.P_{a5}\\
& +\\
& {}^{r_5}\backslash^{\Box}_{\overline{cAMP_i}}\backslash^{\overline{cAMP}_o}_{\Box}\backslash^{\Box}_{r_6}\backslash^{p_5}_{\Box}\backslash_{p_6}.P_{a5} \ + \ {}^{r_5}\backslash^{\Box}_{\overline{CAP_i}}\backslash^{\overline{CAP}_o}_{\Box}\backslash^{\Box}_{r_6}\backslash^{p_5}_{\Box}\backslash_{p_6}.P_{a5}\\
P_{a6} \triangleq\ & {}^{r_6}\backslash^{\Box}_{lac_i}\backslash^{lac_o}_{\Box}\backslash^{\Box}_{cAMP_i}\backslash^{cAMP_o}_{\Box}\backslash^{\Box}_{\overline{I\text{-}OP_i}}\backslash^{\overline{I\text{-}OP}_o}_{\Box}\backslash^{\Box}_{cxt_1}\backslash^{p_6}_{\Box}\backslash^{\Box}_{\tilde{z}_i}\backslash^{\tilde{z}_o}_{\Box}\backslash^{\Box}_{\tilde{y}_i}\backslash^{\tilde{y}_o}_{\Box}\backslash^{\Box}_{\tilde{A}_i}\backslash^{\tilde{A}_o}_{\Box}\backslash_{\tau}.P_{a6}\\
& +\\
& {}^{r_6}\backslash^{\Box}_{I\text{-}OP_i}\backslash^{I\text{-}OP_o}_{\Box}\backslash^{\Box}_{cxt_1}\backslash^{p_6}_{\Box}\backslash_{\tau}.P_{a6}\\
& +\\
& {}^{r_6}\backslash^{\Box}_{\overline{lac_i}}\backslash^{\overline{lac}_o}_{\Box}\backslash^{\Box}_{cxt_1}\backslash^{p_6}_{\Box}\backslash_{\tau}.P_{a6} \ + \ \backslash^{\Box}_{\overline{cAMP_i}}\backslash^{\overline{cAMP}_o}_{\Box}\backslash^{\Box}_{cxt_1}\backslash^{p_6}_{\Box}\backslash_{\tau}.P_{a6}
\end{aligned}$$

Processes for the entities. We exploit the specificity of the example in hand to optimising the code, and we specify exactly the number of solid links that each process encoding an entity must offer. For the always present entities we let:

$$\begin{aligned}
P_{cya} &\triangleq {}^{cya_i}\backslash_{cya_o}.P_{cya} & \qquad P_{crp} &\triangleq {}^{crp_i}\backslash_{crp_o}.P_{crp}\\
P_{lacI} &\triangleq {}^{lacI_i}\backslash_{lacI_o}.P_{lacI} & \qquad P_{lac} &\triangleq {}^{lac_i}\backslash_{lac_o}.P_{lac}
\end{aligned}$$

For the entities always produced (i.e. not present only at the first step), we provide a parametric definition $P_e(s) \triangleq {}^{s_i}\backslash^{\Box}_{s_o}\backslash^{\tilde{s}_i}_{\Box}\backslash_{\tilde{s}_o}.P_e(s) + {}^{\tilde{s}_i}\backslash_{\tilde{s}_o}.P_e(s)$.
There are three entities of the second type:

$$P_{cAMP} \triangleq P_e(cAMP) \qquad P_{CAP} \triangleq P_e(CAP) \qquad P_I \triangleq P_e(I).$$

The entity *I-OP* can be either produced (by a_2) or tested for absence (by a_6). Correspondingly, the process $P_{I\text{-}OP}$ is defined as follows:

$$\begin{aligned}
P_{I\text{-}OP} &\triangleq \textstyle\sum_{h=0}^{1}({}^{I\text{-}OP_i}\backslash^{\Box}_{I\text{-}OP_o}\backslash_{\Box})^h\, {}^{\widetilde{I\text{-}OP}_i}\backslash_{\widetilde{I\text{-}OP}_o}.P_{I\text{-}OP} \ + \ {}^{I\text{-}OP_i}\backslash_{I\text{-}OP_o}.\overline{P_{I\text{-}OP}}\\
\overline{P_{I\text{-}OP}} &\triangleq \textstyle\sum_{h=0}^{1}({}^{\overline{I\text{-}OP}_i}\backslash^{\Box}_{\overline{I\text{-}OP_o}}\backslash_{\Box})^h\, {}^{\widetilde{I\text{-}OP}_i}\backslash_{\widetilde{I\text{-}OP}_o}.P_{I\text{-}OP} \ + \ {}^{\overline{I\text{-}OP}_i}\backslash_{\overline{I\text{-}OP_o}}.\overline{P_{I\text{-}OP}}
\end{aligned}$$

The process $P_{cAMP\text{-}CAP}$ is similar to $P_{I\text{-}OP}$, as it is produced by a_5 and tested for presence by a_6. Its code is in Table 2, in the Appendix. The *lactose* is provided by the context and tested for absence by a_2.

$$
\begin{aligned}
P_{lactose} \triangleq\; & \textstyle\sum_{h=0}^{1}\big({}^{lactose_i}\backslash^{\square}_{lactose_o}\backslash_{\square}\big)^h\; {}^{\widehat{lactose}_i}\backslash_{\widehat{lactose}_o}.P_{lactose} \\
& + \\
& \textstyle\sum_{h=0}^{1}\big({}^{lactose_i}\backslash^{\square}_{lactose_o}\backslash_{\square}\big)^h\; {}^{\underline{lactose_i}}\backslash_{\underline{lactose_o}}.\overline{P_{lactose}} \\
\overline{P_{lactose}} \triangleq\; & \textstyle\sum_{h=0}^{1}\big({}^{\overline{lactose_i}}\backslash^{\square}_{\overline{lactose_o}}\backslash_{\square}\big)^h\; {}^{\widehat{lactose}_i}\backslash_{\widehat{lactose}_o}.P_{lactose} \\
& + \\
& \textstyle\sum_{h=0}^{1}\big({}^{\overline{lactose_i}}\backslash^{\square}_{\overline{lactose_o}}\backslash_{\square}\big)^h\; {}^{\underline{lactose_i}}\backslash_{\underline{lactose_o}}.\overline{P_{lactose}}
\end{aligned}
$$

The process $P_{glucose}$ is similar to $P_{lactose}$ and tested for absence by a_5. Its code is in Table 3, in the Appendix. The entity z can only be produced by rule a_6, while it is never provided by the context. Moreover, there is no rule for testing its presence or absence.

$$
P_z \triangleq {}^{\tilde{z}_i}\backslash^{\square}_{\tilde{z}_o}\backslash^{z_i}_{\square}\backslash_{\underline{z_o}}.P_z \;+\; {}^{\underline{z_i}}\backslash_{\underline{z_o}}.\overline{P_z} \qquad\qquad \overline{P_z} \triangleq {}^{\tilde{z}_i}\backslash^{\square}_{\tilde{z}_o}\backslash^{z_i}_{\square}\backslash_{\underline{z_o}}.\overline{P_z} \;+\; {}^{\underline{z_i}}\backslash_{\underline{z_o}}.\overline{P_z}
$$

The entities y and A are treated in the same way as z. Their processes are in Table 4 in the Appendix.

Context. The entities in DC are assumed always present by default, so no context process is needed for them. The entities z, y, and A are assumed never provided by the context. Their processes are

$$
\begin{aligned}
& Cxt_z \triangleq {}^{cxt_1}\backslash^{\square}_{\underline{z}_i}\backslash^{\underline{z}_o}_{\square}\backslash_{cxt_2}.Cxt_z \qquad\qquad Cxt_y \triangleq {}^{cxt_2}\backslash^{\square}_{\underline{y}_i}\backslash^{\underline{y}_o}_{\square}\backslash_{cxt_3}.Cxt_y \\
& Cxt_A \triangleq {}^{cxt_3}\backslash^{\square}_{\underline{A}_i}\backslash^{\underline{A}_o}_{\square}\backslash_{cxt_4}.Cxt_A
\end{aligned}
$$

For the sake of presentation, we assume that the *lactose* is always provided by the context, in contrast, *glucose* is never provided.

$$
\begin{aligned}
Cxt_{lactose} &\triangleq {}^{cxt_4}\backslash^{\square}_{\widehat{lactose}_i}\backslash^{\widehat{lactose}_o}_{\square}\backslash_{cxt_5}.Cxt_{lactose} \\
Cxt_{glucose} &\triangleq {}^{cxt_5}\backslash^{\square}_{\underline{glucose}_i}\backslash^{\underline{glucose}_o}_{\square}\backslash_{p_1}.Cxt_{glucose}
\end{aligned}
$$

In the following we let $CXT \triangleq Cxt_z|Cxt_y|Cxt_A|Cxt_{lactose}|Cxt_{glucose}$ be the processes for context. The whole system is as follows:

$$
lacOp \triangleq (\nu\, names)(\mathit{\Pi}_{i=1}^{6}P_{ai}|\mathit{\Pi}_{s\in DC}P_s|\mathit{\Pi}_{s\in S\setminus DC}\overline{P}_s|CXT)
$$

Execution. Now, we show two transitions. After the first transition the entity *cAMP-CAP* is produced due to the absence of *glucose*, while the presence of *lactose* inhibits the production of *I-OP*: $lacOp \xrightarrow{(\nu\, names)v} lacOp'$ where:

$$
\begin{aligned}
& v = {}^{\tau}\backslash_{lac_i}\cdots{}^{lactose_o}\backslash_{r3}\cdots{}^{r_5}\backslash_{cAMP_i}\cdots{}^{\overline{glucose}_o}\backslash_{r_6}\cdots{}^{p_5}\backslash_{\widetilde{cAMP\text{-}CAP}_i}\cdots{}^{p_6}\backslash_{\tau} \\
& lacOp' \triangleq (\nu\, names)(\mathit{\Pi}_{i=1}^{6}P_{ai}|\mathit{\Pi}_{s\in AP}P_s|\mathit{\Pi}_{s\in S\setminus AP}\overline{P}_s|CXT)
\end{aligned}
$$

with $AP = DC \cup \{cAMP\text{-}CAP\}$ the actual context.
After the second step the entities z, y and A are produced, due to the presence of $cAMP\text{-}CAP$ and the absence of $I\text{-}OP$, thus $lacOp' \xrightarrow{(\nu\, names)v'} lacOp''$ where:

$$v' = {}^{\tau}\backslash_{lac_i} \cdots {}^{lac_o}\backslash_{cAMP-CAP_i} \cdots {}^{\overline{I-OP}_o}\backslash^{\cdots}_{cxt_1} {}^{\underline{glucose}_o}\backslash^{p_1}_{p_1}\backslash^{p_6}_{\cdots}\backslash^{\tilde{z}_i}_{\tilde{z}_i}\backslash^{\tilde{z}_o}_{\tilde{z}_o}\backslash^{\tilde{y}_i}_{\tilde{y}_i}\backslash^{\tilde{y}_o}_{\tilde{y}_o}\backslash^{\tilde{A}_i}_{\tilde{A}_i}\backslash^{\tilde{A}_o}_{\tilde{A}_o}\backslash_{\tau}$$

$lacOp'' \triangleq (\nu\, names)(\Pi_{i=1}^{6} P_{ai} | \Pi_{s \in AP'} P_s | \Pi_{s \in S \setminus AP'} \overline{P}_s | CXT)$

with $AP' = DC \cup \{z, y, A\}$.

6 Bio-simulation

The classical notion of bisimulation for process algebra equates two processes when one process can simulate all the instructions executed by the other one and viceversa. In its weak formulation, internal instructions, i.e. non visible by external observers, are abstracted away. There are many variants of the bisimulation for process algebras, for example the barbed bisimulation [19] only consider the execution of invisible actions, and then equates two processes when the expose the same prefixes; for the mobile ambients [13], a process algebra equipped with a reduction semantics, a notion of behavioural equivalence equates two processes when they expose the same ambients[17].

There are some previous works based on bisimulation applied to models for biological systems. Barbuti et al [4] define a classical setting for bisimulation for two formalisms: the Calculus of Looping Sequences, which is a rewriting system, and the Brane Calculi, which is based on process calculi. Bisimulation is used to verify properties of the regulation of lactose degradation in Escherichia coli and the EGF signalling pathway. These calculi allow the authors to model membranes' behaviour. [14] presents two quantitative behavioral equivalences over species of a chemical reaction network with semantics based on ordinary differential equations. Bisimulation identifies a partition where each equivalence class represents the exact sum of the concentrations of the species belonging to that class. Bisimulation also relates species that have identical solutions at all time points when starting from the same initial conditions. Both the mentioned formalisms [4,14] adopt a classical approach to bisimulation. Moreover our formalisms is different, for different applications and purposes.

Albeit the bisimulation is a powerful tool for verifying if the behaviour of two different software programs is indistinguishable, in the case of biological systems the classical bisimulation seems to be inappropriate, as it considers too many details. In fact, in a biological soup, a high number of interactions occur every seconds, and generally, biologists are only interested to analyse a small subset of them.

For this reason, we propose an alternative notion of bisimulation, that hereafter we call biosimulation, that allows us to compare two biological systems by restricting the observation to only the limited events of interest.

The transition labels of our systems record detailed information about all the reactions that have been applied in one transition, about the elements that acted as reagents, as inhibitors or that have been produced, or that have been provided by the context.

In a way, we want to query our transition labels to get only the information we are interested in. To this goal, we introduce a simple language that allows us to formulate detailed and partial queries about what happened in a single transition.

First, we need to introduce the $flat$ function that takes a link chain and returns a simple string by eliminating all the channel matching pairs leaving just one channel per each pair:

$$flat(\epsilon) \triangleq \epsilon \qquad flat({}^{\tau}\backslash_{\beta}) \triangleq \beta \qquad flat({}^{\alpha}\backslash_{\tau}) \triangleq \epsilon \qquad flat({}^{\alpha}\backslash_{\beta}) \triangleq \beta$$
$$flat({}^{\alpha}\backslash_{\beta}\upsilon) \triangleq \beta :: flat(\upsilon)$$

where :: is the string concatenation operator.
Now, we can introduce an assertion language that operates on simple strings and that combines regular expression operators with the classical *and* and *or* logical operators.

Definition 11 (Assertion language). *Assertions are built from:*

$$\begin{array}{lcl} \zeta' & ::= & \epsilon \;\mid\; \zeta :: \zeta' \\ re & ::= & . \;\mid\; (\zeta\text{-}\zeta) \;\mid\; \zeta' \;\mid\; re :: re \\ re' & ::= & [\zeta \ldots \zeta] \;\mid\; [\hat{\ } re] \;\mid\; re* \;\mid\; re+ \\ F & ::= & re' \;\mid\; F \vee F \;\mid\; F \wedge F \end{array}$$

where $\zeta \in \{s, \tilde{s}, \hat{s}, \overline{s}, \underline{s}\}$ *with* $s \in \mathcal{C}$.

ζ' is the set of strings composed by characters in the set $\mathcal{C}$. Then, the dot "." stands for any character, ζ_1-ζ_2 is a shorthand for the string starting with ζ_1 and ending with ζ_2 and that in the middle includes all the characters following the alphanumeric order. The square brackets match one of the characters they contain. The symbol $[\hat{\ } \ldots]$, inside the square brackets excludes any of the characters they contain. The star $*$ allows the previous character to be replicated zero or more times. The plus symbol $+$ allows the previous character to be replicated at least one or more times. Then, we can combine the patterns build with the above constructors with the logic operators $\vee$ and $\wedge$.

Definition 12 (Semantics). *Let* υ *be a transition label, and* F *be an assertion. We define* $\upsilon \models F$ *(read as the transition label* υ *satisfies the formula* F*) as:*

- $\upsilon \models \zeta$ *iff* $\zeta \in flat(\upsilon)$.
- $\upsilon \models F_1 \wedge F_2$ *iff* $\upsilon \models F_1 \wedge \upsilon \models F_2$.
- $\upsilon \models F_1 \vee F_2$ *iff* $\upsilon \models F_1 \vee \upsilon \models F_2$.

If it is not the case that $\upsilon \models F$*, then we say that* F *does not hold at* υ *and we write* $\upsilon \not\models F$.

Our language allows us to check if the simulation of a system (i.e. the transition labels, or in other words the trace of a simulation) satisfies the properties of the following types:

1. Has the entity s been used by rule r as reagent ?
2. Has the entity s blocked the application of rule r ?
3. Has the entity s been produced by some rule ?
4. Has the entity s been produced by rule p ?
5. Has the entity s been provided by the context ?
6. Has the rule i been applied ?

The corresponding formulas are as follows:

1. $r[s1 \dots sn] * s[s1 \dots sn]*$
2. $r[s\overline{s}]r'$
3. $\tilde{s}$
4. $p[\tilde{s1} \dots \tilde{sn}] * \tilde{s}[\tilde{s1} \dots \tilde{sn}]*$
5. $\hat{s}$
6. $DADECIDEREcome$

where r' is next rule.

Now, we introduce a slight modified version of the Hennessy Milner Logic (the Hennessy Milner link logic, HM-linkL) []; due to the reasons we explained above, we do not want to look at the complete transition labels, thus we rely on our simple assertion language:

Definition 13 (HM-linkL). *Let F be an assertion, then the set of HM-linkL formulae $\mathcal{G}$are built by the following syntax:*

$$G, H ::= \mathtt{t} \mid \mathtt{f} \mid G \wedge H \mid G \vee H \mid \langle F \rangle G \mid [F]G$$

Definition 14 (semantics of G). *We define $[\![G]\!] \subseteq \mathcal{P}$ by induction on the structure of $G \in \mathcal{G}$:*

$$\begin{array}{ll} [\![\mathtt{t}]\!] \triangleq \mathcal{P} & \quad [\![G \wedge H]\!] \triangleq [\![G]\!] \cap [\![H]\!] \\ [\![\mathtt{f}]\!] \triangleq \emptyset & \quad [\![G \vee H]\!] \triangleq [\![G]\!] \cup [\![H]\!] \end{array}$$

$$\begin{array}{l} [\![\langle F \rangle G]\!] \triangleq \{P \in \mathcal{P} : \exists P', v.P \xrightarrow{v} P' \text{ with } v \models F \text{ and } P' \in [\![G]\!]\} \\ [\![[F]G]\!] \triangleq \{P \in \mathcal{P} : \forall P', v.P \xrightarrow{v} P' \text{ implies } v \models F \text{ and } P' \in [\![G]\!]\} \end{array}$$

We write $P \vdash G$ (P satisfies G) if and only if $P \in [\![G]\!]$.

Please recall that we have defined the behaviour of the contest in a non determinist way, thus at each step, different possible sets of entities can be provided to the system.

Definition 15 (Biosimilarity $\sim_F$). *A* biosimulation $\mathbf{R}_F$, *with respect to the predicate F, is a binary relation over* `link`*-calculus processes such that, if P $\mathbf{R}_F$ Q then:*

$$P \xrightarrow{v} P' \text{ with } v \models F \quad \text{iff} \quad Q \xrightarrow{v'} Q' \text{ with } v' \models F \text{ and } P' \mathbf{R}_F Q'.$$

We let $\sim_F$ denote the largest biosimulation and we say that P is biosimilar *to Q, with respect to F, if $P \sim_F Q$.*

$$\frac{}{E.C \xrightarrow{E,(\emptyset,\emptyset,\emptyset)} C}\ (\mathit{Cxt}) \qquad \frac{}{\{e\} \xrightarrow{\{e\},(\emptyset,\emptyset,\emptyset)} \emptyset}\ (\mathit{Ent})$$

$$\frac{}{(R,I,P) \xrightarrow{\emptyset,(R,I,P)} (R,I,P)|P}\ (\mathit{Pro})$$

$$\frac{e \in I}{(R,I,P) \xrightarrow{\emptyset,(\{e\},\emptyset,\emptyset)} (R,I,P)}\ (\mathit{Hin1})$$

$$\frac{e \in R}{(R,I,P) \xrightarrow{\emptyset,(\emptyset,\{e\},\emptyset)} (R,I,P)}\ (\mathit{Hin2})$$

$$\frac{K_1 \xrightarrow{E_1,(R_1,I_1,P_1)} K_1' \qquad K_2 \xrightarrow{E_2,(R_2,I_2,P_2)} K_2' \qquad (E_1 \cup E_2 \cup R_1 \cup R_2) \cap (I_1 \cup I_2) = \emptyset}{K_1|K_2 \xrightarrow{E_1 \cup E_2,(R_1 \cup R_2, I_1 \cup I_2, P_1 \cup P_2)} K_1'|K_2'}\ (\mathit{Rel})$$

$$\frac{K \xrightarrow{E,(R,I,P)} K' \qquad R \subseteq E}{[K] \xrightarrow{E,(R,I,P)} [K']}\ (\mathit{Sys})$$

Fig. 3: SOS semantics of the reaction system configurations.

7 Labelled Transition System for Reaction Systems

Definition 16.

$$\begin{aligned} S &::= [K] \\ K &::= (R,I,P) \ \mid\ \{e\} \ \mid\ K|K \ \mid C \\ C &::= \mathit{rec}X.C \ \mid\ X \ \mid\ E.C \ \mid\ C + C \end{aligned}$$

We assume that $\{e_1\}|\{e_2\} = \{e_1, e_2\}$

8 Enhanced Reaction Systems

Our encoding increases the expressivity of RS concerning: the behaviour of the context, the possibility of alternative behaviour of mutated entities, the communication between two different reaction systems. It is important to note that our encoding guarantees that from each state, in the cCNA transition system, only one transition comes out, as the dynamics is totally deterministic.

8.1 Recursive contexts

In RS, the behaviour of the context is finite. For the first n steps, it is specified which are the entities that are provided from the context. Using cCNA we can describe in a natural way the behaviour of the context in a recursive way. Then,

$rs1$	$rs2$
$a_1 = (s,\ \ , x)$	$a_2 = (y,\ \ , s)$

Table 1: The two reaction systems $rs1$ and $rs2$.

the context behaviour would not necessarily end after n steps, and could be infinite. For example, in an extended interactive process, we may want that the entity s is intermittently provided by the context every two steps:

$$\begin{array}{lll}
Cxt_s & \triangleq {}^{cxt_j}\backslash^{\square}_{\hat{s}_i}\backslash^{\hat{s}_o}_{\square}\backslash_{cxt_{j+1}}.Cxt_{s'}, & \text{context provides } s;\\
Cxt_{s'} & \triangleq {}^{cxt_j}\backslash^{\square}_{\underline{s}_i}\backslash^{\underline{s}_o}_{\square}\backslash_{cxt_{j+1}}.Cxt_{s''}, & \text{context doesn't provide } s;\\
Cxt_{s''} & \triangleq {}^{cxt_j}\backslash^{\square}_{\underline{s}_i}\backslash^{\underline{s}_o}_{\square}\backslash_{cxt_{j+1}}.Cxt_{s}, & \text{context doesn't provide } s.
\end{array}$$

8.2 Mutating entities

In RS, when an entity is present, it can potentially be involved in each reactions where it is required. With a few more lines of code, in cCNA it is possible to describe the behaviour of a mutation of an entity, in a way that the mutated version of the entity can take part to only a subset of the rules requiring the *normal version* of the entity. For example, let us assume that entity $s1$ is consumed by reactions $a1$ and $a2$. Reaction $a1$ produces also $s1$ if $s2$ is present, otherwise $a1$ produces a mutated version of $s1$, say $s1'$. When $s1'$ is produced, reaction $a2$ behaves in the same way as if $s1$ would be absent, whereas $a2$ recognises the presence of $s1'$ and behaves in the same way as if $s1$ would be present. Technically, in both cases it is enough to add one more non deterministic choice in the code of P_{a1} and P_{a2}.

8.3 Communicating reaction systems

We sketch how it is possible to program two RSs encodings, in a way that the entities that usually come from context of one RS will be provided instead from the other RS.

Example 2. Let $rs1$ and $rs2$ be two RSs, defined by the rules in Table 1.
Now, we set our example such that the two contexts, for $rs1$ and $rs2$, do not provide any entities. We also assume that entity s in $rs1$ is provided by $rs2$, as $rs2$ produces a quantity of s that is enough for $rs1$ and $rs2$. For technical reasons, we can not use the same name for s in both the two RSs, then we use the name ss in $rs2$. We need to modify our translation technique to suite this new setting. As we do not model contexts, we introduce *dummy* channel names dx and dss to model the absence of entities. Also, thanks to the simplicity of the example, we can leave out the use of the p_i channels. This streamlining does not affect the programming technique we propose to make two RSs communicate. First, we translate the reaction in $rs1$:

$$[\![a_1]\!] \triangleq P_{a_1} \triangleq {}^{\tau}\backslash^{\square}_{s_i}\backslash^{s_o}_{\square}\backslash^{\square}_{\tilde{x}_i}\backslash^{\tilde{x}_o}_{\square}\backslash_{a_2}.P_{a_1} + {}^{\tau}\backslash^{\square}_{\overline{s}_i}\backslash^{\overline{s}_o}_{\square}\backslash^{\square}_{dx_i}\backslash^{dx_o}_{\square}\backslash_{a_2}.P_{a_1}$$

Please note, that prefixes of process P_{a_1} end with the channel name a_2, as the link chain is now connected with the reaction of $rs2$. The translation for the entities follows.

$$\llbracket s \rrbracket \triangleq \begin{array}{l} P_s \triangleq {}^{s_i}\backslash^{\square}_{s_o}\backslash^{\hat{s}_i}_{\square}\backslash_{\hat{s}_o}.P_s + {}^{s_i}\backslash_{s_o}.\overline{P_s} \\ \overline{P_s} \triangleq {}^{\overline{s}_i}\backslash^{\square}_{\overline{s}_o}\backslash^{\hat{s}_i}_{\square}\backslash_{\hat{s}_o}.P_s + {}^{\overline{s}_i}\backslash_{\overline{s}_i}.\overline{P_s} \end{array} \qquad \llbracket x \rrbracket \triangleq \begin{array}{l} P_x \triangleq {}^{\tilde{x}_i}\backslash_{\tilde{x}_o}.P_x + {}^{dx_i}\backslash_{dx_o}.\overline{P_x} \\ \overline{P_x} \triangleq {}^{\tilde{x}_i}\backslash_{\tilde{x}_o}.P_x + {}^{dx_i}\backslash_{dx_o}.\overline{P_x} \end{array}$$

The translation for the $rs2$ follows.

$$\llbracket a_2 \rrbracket \triangleq P_{a_2} \triangleq {}^{a_2}\backslash^{\square}_{y_i}\backslash^{y_o}_{\square}\backslash^{\square}_{\tilde{ss}_i}\backslash^{\tilde{ss}_o}_{\square}\backslash_{\tau}.P_{a_2} + {}^{a_2}\backslash^{\square}_{\overline{y}_i}\backslash^{\overline{y}_o}_{\square}\backslash^{\square}_{dss_i}\backslash^{dss_o}_{\square}\backslash_{\tau}.P_{a_2}$$

In the translation of the entities in $rs2$, we introduce the mechanism that allows the entity s (ss in $rs2$) to be provided in $rs1$. Every time ss is produced in $rs2$, a virtual link is created to synchronise with $rs1$ on link ${}^{\hat{s}_i}\backslash_{\hat{s}_o}$:

$$\llbracket ss \rrbracket \triangleq \begin{array}{l} P_{ss} \triangleq {}^{\tilde{ss}_i}\backslash^{\square}_{\hat{s}_i}\backslash^{\hat{s}_o}_{\square}\backslash_{\tilde{ss}_o}.P_{ss} + {}^{dss_i}\backslash_{dss_o}.\overline{P_{ss}} \\ \overline{P_{ss}} \triangleq {}^{\tilde{ss}_i}\backslash^{\square}_{\hat{s}_i}\backslash^{\hat{s}_o}_{\square}\backslash_{\tilde{ss}_o}.P_{ss} + {}^{dss_i}\backslash_{dss_o}.\overline{P_{ss}} \end{array} \qquad \llbracket y \rrbracket \triangleq \begin{array}{l} P_y \triangleq {}^{y_i}\backslash_{y_o}.\overline{P_y} \\ \overline{P_y} \triangleq {}^{\overline{y}_i}\backslash_{\overline{y}_o}.\overline{P_y} \end{array}$$

We now assume that the initial system is $S \triangleq (\nu\, names)(P_{a_1}|P_{a_2}|P_s|P_y|\overline{P_x}|\overline{P_{ss}})$, i.e. only entities s and y are present. Now, the only possible transition has the following label (that we report without restriction):

$$\tau\backslash^{s_i}_{s_i}\backslash^{s_o}_{s_o}\backslash^{\tilde{x}_i}_{\tilde{x}_i}\backslash^{\tilde{x}_o}_{\tilde{x}_o}\backslash^{a_2}_{a_2}\backslash^{y_i}_{y_i}\backslash^{y_o}_{y_o}\backslash^{\tilde{ss}_i}_{\tilde{ss}_i}\backslash^{\hat{s}_i}_{\hat{s}_i}\backslash^{\hat{s}_o}_{\hat{s}_o}\backslash^{\tilde{ss}_o}_{\tilde{ss}_o}\backslash\tau,$$

where the black links belong to the prefixes of P_{a_1}, and P_{a_2}, the blue links belong to P_s, the gray links belong to P_y, and $\overline{P_x}$ and the red links belong to $\overline{P_{ss}}$. After the execution, the entity s is still present in $rs1$ as it has been provided by $rs2$.

As we have briefly sketched, our model of two *communicating reaction systems* can enable the study of the behaviour of one RS in relation to another one. In Example 2 describing the behaviour of the *lac* operon, the two entities lactose and glucose are controlled non deterministically by the context. In our framework, instead, by exploiting the expressivity of cCNA, the *lac* operon system has been connected with the two systems producing the lactose and the glucose. This way, the presence of these two entities in the *lac* operon system can be regulated by realistic mechanisms.

9 Conclusion

In this paper we have introduced a variant of the `link`-calculus where prefixes are link chains and no more single links, as it was briefly described in the future work section in [7]. This variant allowed us to define an elegant embedding of reaction systems, an emerging formalism to model computationally biochemical systems. This translation shows several benefits. For instance, the context behaviour can also be expressed recursively. Entity mutations can be expressed easily. Reaction systems can communicate between them.

We believe that our embedding can contribute to extend the applications of reaction systems to diverse fields of computer science, and life sciences. As we have already mentioned, the evolution of each process resulting from our embedding is deterministic, thus we do not have the problem of having infinitely many transitions in the produced labelled transition system. In any case, we can exploit the implementation of the symbolic semantics of the `link`-calculus [12] that can be found in [21].

As future work, we plan to implement a prototype of our framework, with an automatic translation from RSs to `link`-calculus. We believe that our work can also help to extend the framework of RSs towards a model which can improve the communication between different RSs. We also believe that our work can make possible to investigate how to apply formal techniques to prove properties of the modeled systems [15,22,9].

References

1. S. Azimi. Steady states of constrained reaction systems. *Theor. Comput. Sci.*, 701(C):20–26, 2017.
2. S. Azimi, B. Iancu, and I. Petre. Reaction system models for the heat shock response. *Fundamenta Informaticae*, 131(3-4):299–312, 2014.
3. R. Barbuti, R. Gori, F. Levi, and P. Milazzo. Investigating dynamic causalities in reaction systems. *Theor. Comput. Sci.*, 623:114–145, 2016.
4. R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. Bisimulations in calculi modelling membranes. *Formal Aspects of Computing*, 20(4):351–377, 2008.
5. A. Bernini, L. Brodo, P. Degano, M. Falaschi, and D. Hermith. Process calculi for biological processes. *Natural Computing*, 17(2):345–373, 2018.
6. C. Bodei, L. Brodo, and R. Bruni. Open multiparty interaction. In *Recent Trends in Algebraic Development Techniques, 21st International Workshop, WADT 2012*, volume 7841 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2012.
7. C. Bodei, L. Brodo, and R. Bruni. A formal approach to open multiparty interactions. *Theoretical Computer Science*, 763:38–65, 2019.
8. C. Bodei, L. Brodo, R. Bruni, and D. Chiarugi. A flat process calculus for nested membrane interactions. *Sci. Ann. Comp. Sci.*, 24(1):91–136, 2014.
9. C. Bodei, L. Brodo, R. Gori, F. Levi, A. Bernini, and D. Hermith. A static analysis for Brane Calculi providing global occurrence counting information. *Theoretical Computer Science*, 696:11–51, 2017.
10. R. Brijder, A. Ehrenfeucht, M. Main, and G. Rozenberg. A tour of reaction systems. *International Journal of Foundations of Computer Science*, 22(07):1499–1517, 2011.
11. L. Brodo. On the expressiveness of pi-calculus for encoding mobile ambients. *Mathematical Structures in Computer Science*, 28(2):202–240, 2018.
12. L. Brodo and C. Olarte. Symbolic semantics for multiparty interactions in the link-calculus. In *Proc. of SOFSEM'17*, volume 10139 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2017.
13. L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

14. L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Forward and backward bisimulations for chemical reaction networks. In *26th International Conference on Concurrency Theory, CONCUR 2015*, volume 42, pages 226–239. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2015.
15. D. Chiarugi, M. Falaschi, D. Hermith, C. Olarte, and L. Torella. Modelling non-markovian dynamics in biochemical reactions. *BMC Systems Biology*, 9(S-3):S8, 2015.
16. L. Corolli, C. Maj, F. Marinia, D. Besozzi, and G. Mauri. An excursion in reaction systems: From computer science to biology. *Theoretical Computer Science*, 454:95–108, 2012.
17. A. Gordon and L. Cardelli. Equational properties of mobile ambients. *Mathematical Structures in Computer Science*, 13(3):371–408, june 2003.
18. A. Męski, W. Penczek, and G. Rozenberg. Model checking temporal properties of reaction systems. *Information Sciences*, 313:22–42, 2015.
19. R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Automata, Languages and Programming*, pages 685–695, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
20. F. Okubo and T. Yokomori. The computational capability of chemical reaction automata. *Natural Computing*, 15(2):215–224, 2016.
21. C. Olarte. SiLVer: Symbolic links verifier, Dec. 2018. `http://subsell.logic.at/links/links-web/index.html`.
22. C. Olarte, D. Chiarugi, M. Falaschi, and D. Hermith. A proof theoretic view of spatial and temporal dependencies in biochemical systems. *Theor. Comput. Sci.*, 641:25–42, 2016.

A Omitted Proofs

In this section we report the proofs for the results in Section 4.

Lemma 1. *Let $\mathcal{A} = (S, A)$ be a RS and let $\pi = (\gamma, \delta)$ be an extended interactive process in $\mathcal{A}$. Let $P = [\![\mathcal{A}, \gamma]\!]$ its cCNA translation. If exists P' such that $t = (P \xrightarrow{(\nu\, names)\upsilon} P')$ is a transition of P, then*

1. *for each reaction $a_j \in A$, the corresponding channels r_j and p_j appear in υ; for each entity $s_h \in S$ (where h is the identifying number of s), the corresponding channel s_h (suitably decorated), and the corresponding channel cxt_h appear in υ;*
2. *for each reaction $a \in A$ and each entity $s \in S$, each virtual link offered by processes P_a and Cxt_s is overlapped by exactly one solid link offered by processes representing entities.*

Proof. We prove the two items separately:

1. by Definition 10, all the names r_j, p_j; all the names s_i, s_o, in all their annotated versions, and all the names cxt_h, are restricted. Moreover, the only prefixes that start with τ are those associated to reaction a_1 (as $r_1 = \tau$) and the only prefixes that end with τ are those associated to reaction a_u (as $p_{u+1} = \tau$, where u is the number of reactions). Thus all the prefixes associated with reactions and contexts must be concatenated (remember that $r_{u+1} = cxt_1$ and $cxt_{w+1} = p_1$, where w is the number of entities), forming the backbone of the label. Since all context processes are involved, then all entities processes are also involved. Then, all the processes P_a, P_s (or $\overline{P_s}$), and Cxt_h must participate to each transition.
2. for each reaction $aj \in A$, the cCNA code of P_{aj} leaves one virtual link between two solid links of the types ${}^{r_j}\backslash_{\dots}\backslash^{\square}_{s_i}\backslash^{s_o}_{\square}\backslash_{\dots}\backslash^{\dots}_{r_{j+1}}\ {}^{p_j}\backslash_{\dots}\backslash_{p_{j+1}}$. Then, it derives that the process P_s, encoding the behaviour of entity s, can participate by filling the virtual link in the above transition by only offering one solid link of the type ${}^{s_i}\backslash_{s_o}$. In fact, there is no other way to generate a solid chain from s_i to s_o. The same reasoning holds for the processes Cxt_h and for all the annotated versions of s_i, s_o.

Proposition 1 (Correctness 1). *Let $P = [\![\mathcal{A}, \gamma]\!]$ with*
$P = (\nu\, names)(\Pi_{a\in A} P_a | \Pi_{s\in S} Cxt_s | \Pi_{s\in C_0} P_s | \Pi_{s\notin C_0} \overline{P}_s)$.
If there exists P' such that $P \xrightarrow{\upsilon} P'$, it holds that:
$\upsilon = {}^{\tau}\backslash_{\tau} \dots {}^{\tau}\backslash_{\tau}$ and
$P' = (\nu\, names)(\Pi_{a\in A} P_a | \Pi_{s\in S} Cxt_s | \Pi_{s\in C_1\cup D_1} P_s | \Pi_{s\notin C_1\cup D_1} \overline{P}_s)$.
Moreover, given $\pi^1 = (\gamma^1, \delta^1)$, we have $P' = [\![\mathcal{A}, \gamma^1]\!]$.

Proof. First, we note that all the channels in the system are restricted, see Def. 10, then it holds that the transition labels are of the form $\upsilon = {}^{\tau}\backslash_{\tau} \dots {}^{\tau}\backslash_{\tau}$. Now, by Definition 10 and by Lemma 1.1, all the channels r_j, p_j, with $j \in [1, \dots, u]$, and cxt_h, with $h \in [1, \dots, w]$, and all the annotated versions of s_i, s_o

are restricted. Also, processes Cxt_s always requires the interaction with P_s on either on channels $\hat{s}_i$, $\hat{s}_o$ or on channels $\underline{s}_i$, $\underline{s}_o$. It derives that all the processes: P_a (coding the behaviour of reaction $a \in A$), P_s (coding the behaviour of entity $s \in S$), and Cxt_s (coding the behaviour of the context regarding the entity s) have been involved in the transition. We have the following cases:

(a) processes P_a encoding a reaction a, with serial number j, producing the entity s provide a code of this type: $P_a \triangleq {}^{r_j}\backslash_{\dots}\backslash^{\square}_{r_{j+1}}\backslash^{p_j}_{\square}\backslash_{\dots}\backslash^{\square}_{\tilde{s}_i}\backslash^{\dots\tilde{s}_o}_{\square}\backslash_{\dots}\backslash_{p_{j+1}}.P_a$;
(b) processes P_a encoding a reaction a, with serial number j, consuming the entity s provide a code of this type: $P_a \triangleq {}^{r_j}\backslash_{\dots}\backslash^{\square}_{s_i}\backslash_{\square}{}^{s_o}\backslash_{\dots}\backslash^{\square}_{r_{j+1}}\backslash^{p_j}_{\square}\backslash_{\dots}\backslash_{p_{j+1}}.P_a$;
(c) processes P_a encoding a reaction a, with serial number j, requiring the absence of the entity s provide a code of this type:
$P_a \triangleq {}^{r_j}\backslash_{\dots}\backslash^{\square}_{\bar{s}_i}\backslash_{\square}{}^{\bar{s}_o}\backslash_{\dots}\backslash^{\square}_{r_{j+1}}\backslash^{p_j}_{\square}\backslash_{\dots}\backslash_{p_{j+1}}.P_a$;
(d) processes P_a encoding a reaction a (that cannot be applied), with serial number j, execute a code capturing either the absence of one of its reactants (case 1), or the presence of one of its inhibitors (case 2):
1. $P_a \triangleq {}^{r_j}\backslash_{\dots}\backslash^{\square}_{\bar{s}_i}\backslash_{\square}{}^{\bar{s}_o}\backslash_{\dots}\backslash^{\square}_{r_{j+1}}\backslash^{p_j}_{\square}\backslash_{\dots}\backslash_{p_{j+1}}.P_a$.
2. $P_a \triangleq {}^{r_j}\backslash_{\dots}\backslash^{\square}_{s_i}\backslash_{\square}{}^{s_o}\backslash_{\dots}\backslash^{\square}_{r_{j+1}}\backslash^{p_j}_{\square}\backslash_{\dots}\backslash_{p_{j+1}}.P_a$.

Now, we consider the structure of the process $Cxt_s = Cxt^0_s$. By Definition 10, Cxt^0_s is the unique process encoding the behaviour of the context regulating the supply of s. The code of the process Cxt^t_s, with $t \geq 0$, has the following structure:

(e) $Cxt^i_s \triangleq {}^{cxt_h}\backslash^{\square}_{\hat{s}_i}\backslash^{\hat{s}_o}_{\square}\backslash_{cxt_{h+1}}.Cxt^{i+1}_s$, if $s \in C_{i+1}$;
(f) $Cxt^i_s \triangleq {}^{cxt_h}\backslash^{\square}_{\underline{s}_i}\backslash^{\underline{s}_o}_{\square}\backslash_{cxt_{h+1}}.Cxt^{i+1}_s$, if $s \notin C_{i+1}$;

The code executed by P_s has the following structure:

(g) $P_s \triangleq \sum_{h,k\geq 0}({}^{s_i}\backslash^{\square}_{s_o}\backslash_{\square})^h\, {}^{\hat{s}_i}\backslash^{\square}_{\hat{s}_o}\backslash_{\square}({}^{\tilde{s}_i}\backslash^{\square}_{\tilde{s}_o}\backslash_{\square})^k\, .P_s$, if $s \in C_{i+1}$;
(h) $P_s \triangleq \sum_{h\geq 0,k\geq 1}({}^{s_i}\backslash^{\square}_{s_o}\backslash_{\square})^h\, {}^{\underline{s}_i}\backslash^{\square}_{\underline{s}_o}\backslash_{\square}({}^{\tilde{s}_i}\backslash^{\square}_{\tilde{s}_o}\backslash_{\square})^k.P_s$, if $s \notin C_{i+1}$
(i) $P_s \triangleq \sum_{h\geq 0}({}^{s_i}\backslash^{\square}_{s_o}\backslash_{\square})^h\, {}^{\underline{s}_i}\backslash_{\underline{s}_o}.\overline{P}_s$, if $s \notin C_{i+1}$

where, by Lemma 1.2, h is the number of reactions requiring the presence of s plus possibly some reactions not requiring s; and k is the number of reactions producing s. The code executed by $\overline{P}_s$ has the following structure:

(g') $\overline{P}_s \triangleq \sum_{h,k\geq 0}({}^{\bar{s}_i}\backslash^{\square}_{\bar{s}_o}\backslash_{\square})^h\, {}^{\hat{s}_i}\backslash^{\square}_{\hat{s}_o}\backslash_{\square}({}^{\tilde{s}_i}\backslash^{\square}_{\tilde{s}_o}\backslash_{\square})^k\, .P_s$, if $s \in C_{i+1}$;
(h') $\overline{P}_s \triangleq \sum_{h\geq 0,k\geq 1}({}^{\bar{s}_i}\backslash^{\square}_{\bar{s}_o}\backslash_{\square})^h\, {}^{\underline{s}_i}\backslash^{\square}_{\underline{s}_o}\backslash_{\square}({}^{\tilde{s}_i}\backslash^{\square}_{\tilde{s}_o}\backslash_{\square})^k.P_s$, if $s \notin C_{i+1}$
(i') $\overline{P}_s \triangleq \sum_{h\geq 0}({}^{\bar{s}_i}\backslash^{\square}_{\bar{s}_o}\backslash_{\square})^h\, {}^{\underline{s}_i}\backslash_{\underline{s}_o}.\overline{P}_s$, if $s \notin C_{i+1}$

where, by Lemma 1.2, h is the number of reactions requiring the absence of s plus possibly some reactions requiring s; and k is the number of reactions producing s.

It is worth nothing that, depending on the presence (P_s) or the absence ($\overline{P}_s$) of each entity s, for each process P_a (encoding a reaction a) the choice between the execution of the reaction code (points **(a)**, **(b)**, **(c)**) or the code expressing that reaction a is not applicable (point **(d)**) is deterministic. Also, the building of the code of processes Cxt_s (points **(e)**, **(f)**), follows the evolution of γ. It derives that the trend followed by the processes P_s (or $\overline{P}_s$) is also deterministic (points **(g)**, **(h)**, **(i)** or **(g')**, **(h')**, **(i')**), leading to $P' = [\![\mathcal{A}, \gamma^1]\!]$.

Corollary 1 (Correctness 2). *Let* $P = [\![\mathcal{A}, \gamma]\!]$ *and* $j \geq 1$*. If there exists* P'' *such that* $P \xrightarrow{{}^{\tau}\backslash_{\tau}\ldots{}^{\tau}\backslash_{\tau}}{}^{j} P''$*, then letting* $\pi^j = (\gamma^j, \delta^j)$ *we have* $P'' = [\![\mathcal{A}, \gamma^j]\!]$*.*

Proof. We proceed by induction on the transition number $j \geq 0$.
case $j = 1$
This case falls into the case of Proposition 1.
case $j > 1$
By inductive hypothesis, it holds that $\exists$ P' such that $P \xrightarrow{{}^{\tau}\backslash_{\tau}\ldots{}^{\tau}\backslash_{\tau}}{}^{j-1} P'$ and $P' = [\![\mathcal{A}, \gamma^{j-1}]\!]$.
As P' is the encoding of an extended interactive process, by Proposition 1, it exists P'' such that $P' \xrightarrow{{}^{\tau}\backslash_{\tau}\ldots{}^{\tau}\backslash_{\tau}} P''$, and $P'' = [\![\mathcal{A}, \gamma^j]\!]$.

Proposition 2 (Completeness 1). *Let* $P = [\![\mathcal{A}, \gamma]\!]$ *and* $\pi^1 = (\gamma^1, \delta^1)$*. Then,* $P \xrightarrow{{}^{\tau}\backslash_{\tau}\ldots{}^{\tau}\backslash_{\tau}} P' = [\![\mathcal{A}, \gamma^1]\!]$*.*

Proof. By Proposition 1, if there exists P' such that $P \xrightarrow{{}^{\tau}\backslash_{\tau}\ldots{}^{\tau}\backslash_{\tau}} P'$, then the structure of P' is deterministically computed.
Now, to prove that always exists P', we observe that even in the case no reaction a is applicable in the interactive process π in A, then process P can always execute a step transition, as its subprocesses P_a can always execute one of the *alternative code for when reaction a is not applicable* (see Definition 10, code for P_a processes).

Corollary 2 (Completeness 2). *Let* $P = [\![\mathcal{A}, \gamma]\!]$ *and* $\pi^j = (\gamma^j, \delta^j)$*. Then,* $P \xrightarrow{{}^{\tau}\backslash_{\tau}\ldots{}^{\tau}\backslash_{\tau}}{}^{j} P'' = [\![\mathcal{A}, \gamma^j]\!]$*.*

Proof. The proof proceeds by induction on the number j, and it is similar to the one of Corollary 1

B The *lac* operon encoding

$$\begin{aligned}
P_{cAMP\text{-}CAP} \triangleq\ & \textstyle\sum_{h=0}^{1}\left({}^{cAMP\text{-}CAP_i}\backslash^{\square}_{cAMP\text{-}CAP_o}\backslash\square\right)^h\ {}^{cA\widetilde{MP\text{-}C}AP_i}\backslash_{cA\widetilde{MP\text{-}C}AP_o}.P_{cAMP\text{-}CAP}\\
& + \\
& {}^{cAMP\text{-}CAP_i}\backslash_{cAMP\text{-}CAP_o}.\overline{P_{cAMP\text{-}CAP}}\\[1em]
\overline{P_{cAMP\text{-}CAP}} \triangleq\ & \textstyle\sum_{h=0}^{1}\left({}^{\overline{cAMP\text{-}CAP}_i}\backslash^{\square}_{\overline{cAMP\text{-}CAP}_o}\backslash\square\right)^h\ {}^{cA\widetilde{MP\text{-}C}AP_i}\backslash_{cA\widetilde{MP\text{-}C}AP_o}.P_{cAMP\text{-}CAP}\\
& + \\
& {}^{\overline{cAMP\text{-}CAP}_i}\backslash_{\overline{cAMP\text{-}CAP}_o}.\overline{P_{cAMP\text{-}CAP}}
\end{aligned}$$

Table 2: The process for *cAMP-CAP*.

$$\begin{aligned}
P_{glucose} \triangleq\ & \textstyle\sum_{h=0}^{1}\left({}^{glucose_i}\backslash^{\square}_{glucose_o}\backslash\square\right)^h\ {}^{gl\widehat{ucose}_i}\backslash_{gl\widehat{ucose}_o}.P_{glucose}\\
& + \\
& \textstyle\sum_{h=0}^{1}\left({}^{glucose_i}\backslash^{\square}_{glucose_o}\backslash\square\right)^h\ {}^{\underline{glucose_i}}\backslash_{\underline{glucose_o}}.\overline{P_{glucose}}\\[1em]
\overline{P_{glucose}} \triangleq\ & \textstyle\sum_{h=0}^{1}\left({}^{\overline{glucose}_i}\backslash^{\square}_{\overline{glucose}_o}\backslash\square\right)^h\ {}^{gl\widehat{ucose}_i}\backslash_{gl\widehat{ucose}_o}.P_{glucose}\\
& + \\
& \textstyle\sum_{h=0}^{1}\left({}^{\overline{glucose}_i}\backslash^{\square}_{\overline{glucose}_o}\backslash\square\right)^h\ {}^{\underline{glucose_i}}\backslash_{\underline{glucose_o}}.\overline{P_{glucose}}
\end{aligned}$$

Table 3: The process for *glucose*.

$$\begin{array}{ll}
P_A \triangleq {}^{\tilde{A}_i}\backslash^{\square}_{\tilde{A}_o}\backslash^{\underline{A_i}}_{\square}\backslash_{\underline{A_o}}.P_A\ +\ {}^{\underline{A_i}}\backslash_{\underline{A_o}}.\overline{P_A} & \quad P_y \triangleq {}^{\tilde{y}_i}\backslash^{\square}_{\tilde{y}_o}\backslash^{\underline{y_i}}_{\square}\backslash_{\underline{y_o}}.P_y\ +\ {}^{\underline{y_i}}\backslash_{\underline{y_o}}.\overline{P_y}\\
\overline{P_A} \triangleq {}^{\tilde{A}_i}\backslash^{\square}_{\tilde{A}_o}\backslash^{\underline{A_i}}_{\square}\backslash_{\underline{A_o}}.\overline{P_A}\ +\ {}^{\underline{A_i}}\backslash_{\underline{A_o}}.\overline{P_A} & \quad \overline{P_y} \triangleq {}^{\tilde{y}_i}\backslash^{\square}_{\tilde{y}_o}\backslash^{\underline{y_i}}_{\square}\backslash_{\underline{y_o}}.\overline{P_y}\ +\ {}^{\underline{y_i}}\backslash_{\underline{y_o}}.\overline{P_y}
\end{array}$$

Table 4: The processes for y and A.