Formal Molecular Biology Done in CCS-R

Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 180 (2007) 31–49
www.elsevier.com/locate/entcs

Formal Molecular Biology Done in CCS-R

Vincent Danos[1]

Université Paris VII, CNRS
Paris, France

Jean Krivine[2]

INRIA
Rocquencourt, France

Abstract

We present CCS-R, a reversible variant of Milner's CCS offering a backtracking mechanism. Formalization of biological systems satisfying a "perfect mix" assumption within CCS-R is discussed.

Keywords: Reversible computation. CCS, backtracking mechanism

1 A Language for the cell

What some consider a revolution in biology has started a few years ago. New measurement technologies, such as DNA micro-arrays, are now producing huge amounts of data at the molecular level. For the first time in biology, a top-down approach, complementing the usual bottom-up one, is shown to be possible [10]. These genome-wide snapshots could be immensely valuable in the process of reconstruction and validation of the cell molecular mechanisms. But, with this new possibility comes also the very basic question of how what one wants to reconstruct should be represented in the first place. In which language should the dense networks of interacting processes the cell is made of be described, simulated and analyzed?

The bulk of the reconstruction effort for now is aiming at genetic regulation networks, that is graphs tracing the mutual influences of genes, and there isn't really a need of a formalism so far because the situation is simple and clear enough. But sooner or later further levels of details, such as protein-protein interactions

[1] Email: Vincent.Danos@pps.jussieu.fr
[2] Email: Jean.Krivine@inria.fr

doi:10.1016/j.entcs.2004.01.040

and protein-DNA interactions, will have to find their place in the picture. Various static graphical notations are already commonplace in molecular biology, as in the KEGG pathway data-base for instance [12], but reconstruction tasks demand access to the dynamics. The language one needs to represent the fine molecular details of the cellular processes, the "language of the cell" so to speak, has to describe the evolution of these processes.

As one can imagine, the question hasn't gone unnoticed, and many different languages have already been proposed: plain differential systems [18], Petri Nets of various sorts [9,13], Hybrid Automata [8] and State Charts [11] to name a few. All these are giving means to construct models but we remark that they don't, by themselves, propose a picture of what molecular biology is.

1.1 π-calculus modeling.

Recently, Regev, Shapiro [17,15,16] and a number of other authors [1,3] have brought to the fore the idea that process algebras, languages otherwise developed for the modeling of communication in decentralized computational systems, might be useful. This proposition belongs to a markedly different category since, in some sense, it is trying to explain, at a suitable level of idealization, what molecular biology is. For instance, in π-calculus, a molecule is a π-calculus agent, binding is communication, continuous physical contact is private name sharing, etc.

Seducive as this view on molecular interactions may be, if only because these interactions are really asynchronous and concurrent, one has to confess that the actual code representing precise biological systems (such as the EGF-MAPK cascade in Regev's formalization) is sometimes hardly legible and, worse, that there is much latitude in the way a given biological phenomenon is represented.

Moreover, and here we are getting closer to the topic of this paper, the examples that were given so far are always running *forward*, rarely letting a binding or an activation be undone. While this might be a quite reasonable assumption in some cases, for instance when describing a signal transduction pathway, it is certainly a departure from actual biochemistry where practically all reactions are reversible.

1.2 Reversibility and regulation.

Not only most reactions are reversible but reversibility actually impacts on the system behaviour. Fig. 1 illustrates this with a biologically plausible example: proteins A and B are competing to bind with some C (which could be a complex or a group of binding sites on DNA). Since binding is blind and a purely local operation, one may and will reach sometimes an indefinite state where both A and B are bound with C. But the system won't stay there, sooner or later A or B will leave. Now if one thinks of this system as a molecular switch, with A switching the system on and B preventing this, then it is crucial that the system can escape from the intermediate state, or else the switch is off forever.

So reversibility is not just a biochemical constraint bearing on the cell self-programming, it is actually a basic mechanism in regulation that prevents deadlocks

and is instrumental in obtaining switch-like behaviours.

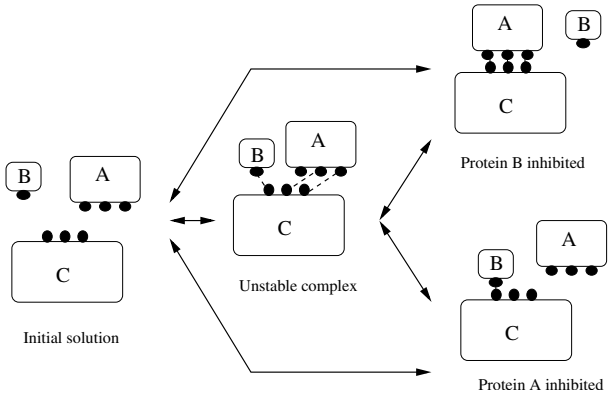

Fig. 1. Competition between two proteins.

The contribution of this paper is to introduce a process algebra with reversibility wired in the syntax.

In principle, reversibility can be expressed in π-calculus, and one might well wonder if reversibility makes a worthwhile addition. We think it does. First, reversibility seems so much the rule in biological systems that it is natural to build it in the calculus and bring in so doing the modeling language closer to real thing. Second, while it is easy to declare that specific actions are irreversible and therefore unbacktrackable, it is not a simple matter to encode reversible actions in a world where irreversibility is the default option. It is certainly possible but has a certain cost in terms of time, legibility and introduces plenty of non biological events in the model. Our reversible language is also relying on a much simpler communication algebra than π, namely CCS, and therefore one can expect our models to perform better on the grounds of legibility and analyzability as well.

1.3 Outline.

The process algebraic development and the biological application are kept in separate sections. For the reader with a biological background, this is as lively an introduction as we could manage to process algebras. Care was taken to explain process algebraic matters in some details and with many examples, but some familiarity with CCS of course would ease the understanding of the paper. We recommend Milner's book [14].

First, we define the basic syntax, give a few examples and prove that backtracking is sound in the sense that CCS and CCS-R systems have the same reachable states. Section 3 discusses the modeling of biomolecular interactions and makes a case for a slight modification of CCS-R, called CCS-Rμ, remedying the fake causal dependencies induced by the expansion law by introducing *multi-actions*. Section 4 illustrates the CCS-Rμ modeling ability under a "perfect mix assumption", by which we mean that all reactants may interact in the solution provided they have a complementary interface. And finally section 5 discusses what happens when the mix isn't perfect.

Acknowledgement

This work owes much to the lectures on biological systems given in 2003 at University Paris 7 by Vincent Schächter; the authors are also happy to thank Fabien Tarissan for fruitful syntactic discussions on CCS-R.

2 CCS-R

As said, our starting point is Milner's CCS [14] which describes interaction as binary synchronized communication. Something happens in a CCS system when two agents are doing complementary actions at the same time, very much as a handshake.

Processes can try some different synchronizations, a behaviour which is written $\sum a_i.P_i$; the sum then represents the willingness of the process to do any of the actions a_i and the P_i are the corresponding subsequent behaviours. Processes can also divide in parallel components, often called threads; this is written $\prod P_i$ where the P_i are the concurrently running sub-processes.

2.1 A syntax for backtrack

The plan to implement backtrack is to assign an identifier to each thread and then an individual memory stack keeping track of past communications. Upon doing a forward transition the information needed for roll-back will have to be stored on the memory stack.

Two constraints are shaping the actual syntactic solution explained below. First comes *soundness*: reversing computations should not give access to formerly unreachable states; the second is *expressiveness*: the memorizing scheme should not induce fake causal dependencies on backward sequences of actions. This section is only concerned with the first question, while the second will be taken care of later in the paper.

The syntax for CCS-R is given in Fig. 2 and defines actions, processes and memories. One records two kinds of events in memories, communications and linking (unfolding of a process definition). The set of identifiers, written $\mathcal{U}$ is taken to be $\mathbb{N}^\star$, that is identifiers are words of integers. We also use the following shorthand notation :

$$\prod M_i[P_i]_{u_i} := M_1[P_1]_{u_1} \mid \ldots \mid M_k[P_k]_{u_k}.$$

It is understood, as in CCS, that each constant K is accompanied by a definition $K := P$ where P is a plain CCS process.

Actions:	a	$::= x \mid \bar{x} \mid \ldots$	Action on a channel
		$\mid \tau$	Silent action
Processes:	P	$::= 0$	End of process
		$\mid \sum a_i.P_i$	Choice
		$\mid \prod M_i[P_i]_{u_i}$	Concurrent threads
		$\mid K$	Constant
		$\mid (\nu x)P$	Restriction
Memories:	M	$::= \langle def{:}K(u)\rangle.M$	Linking trace
		$\mid \langle u,a,P\rangle.M$	Communication trace
		$\mid \langle\rangle$	Empty memory

Fig. 2. CCS-R

2.2 From CCS-R to CCS and back.

Our calculus is clearly only a "decoration" of CCS which can be erased by way of the following forgetful map ϕ : CCS-R $\longrightarrow$ CCS:

$$\begin{aligned}
\phi(0) &= 0\\
\phi(\textstyle\sum a_i.P_i) &= \textstyle\sum a_i.\phi(P_i)\\
\phi(\textstyle\prod M_i[P_i]_{u_i}) &= \textstyle\prod \phi(P_i)\\
\phi(K) &= K\\
\phi((\nu x)P) &= (\nu x)\phi(P)
\end{aligned}$$

Conversely one can equip a CCS process with identifiers and empty memories with the following $\mathcal{U}$-indexed family of lifting functions $\ell : \mathcal{U} \times$ CCS $\longrightarrow$ CCS-R:

$$\begin{aligned}
&\ell_u(0) = 0\\
&\ell_u(\textstyle\sum a_i.P_i) = \textstyle\sum a_i.\ell_u(P_i)\\
&\ell_u(\textstyle\prod P_i) = \textstyle\prod \langle\rangle[\ell_{u.i}(P_i)]_{u.i}\\
&\ell_u(K) = K\\
&\ell_u((\nu x)P) = (\nu x)\ell_u(P)
\end{aligned}$$

Let CCS-R$_i \subset$ CCS-R stand for the set of CCS-R terms with empty memories (or in initial state). All lifting maps have their codomains included in CCS-R$_i$ and are post-inverses to the forgetting map ϕ, that is for all $u \in \mathcal{U}$, $\phi \circ \ell_u$ is the identity. Not quite the converse ! The transformation $\ell_u \circ \phi$ is erasing all memories.

2.3 CCS-R structural congruence

We now want to define structural congruence, written $\equiv$. This is giving some flexibility in the handling of the syntax. The first and most important principle is that memory can be distributed or shared among sub-threads:

$$M[\prod \langle\rangle[P_i]_{u_i}]_u \equiv \langle\rangle[\prod M[P_i]_{u_i}]_u \qquad \textbf{(share)}$$

Together with this first equation that will ensure consistency of the memorisation mechanism, we take care of process definition unfolding. When $K := P$ we set (recall P is a plain CCS process):

$$M[K]_u \equiv \langle def{:}K(u)\rangle.M[\ell_u(P)]_u \qquad \textbf{(link)}$$

New syntactic sub-threads have to be labelled "on the fly", for instance, if $K := (x.K + y.0) \mid \bar{x}.K$, then:

$$\begin{aligned} \ell_1((x.K + y.0) \mid \bar{x}.K) &= \langle\rangle[\ell_{11}(x.K + y.0)]_{11} \mid \langle\rangle[\ell_{12}(\bar{x}.K)]_{12} \\ &= \langle\rangle[x.K + y.0]_{11} \mid \langle\rangle[\bar{x}.K]_{12}, \end{aligned}$$

$$\langle\rangle[K]_1 \equiv \langle def{:}K(1)\rangle[\langle\rangle[x.K + y.0]_{11} \mid \langle\rangle[\bar{x}.K]_{12}]_1$$

To conclude the definition we add the usual CCS equations:

$$\begin{aligned} P_1 \mid P_2 &\equiv P_2 \mid P_1 \\ (P_1 \mid P_2) \mid P_3 &\equiv P_1 \mid (P_2 \mid P_3) \\ P_1 + P_2 &\equiv P_2 + P_1 \\ (P_1 + P_2) + P_3 &\equiv P_1 + (P_2 + P_3) \\ P + 0 &\equiv P \\ (\nu x)P_1 \mid P_2 &\equiv (\nu x)(P_1 \mid P_2) \quad \text{if } x \notin P_2 \end{aligned}$$

and define structural congruence as the least equivalence relation on terms closed under all syntactical constructions and containing the equations above.

2.4 Transition rules

With our structural congruence in place, we can now define a labelled transition system (LTS for short) for CCS-R. This is describing the structure of all possible

computation traces.

A process can evolve through two kinds of transitions: internal or τ-transitions, with labels of the form (u, r, τ) where τ is the silent action and u, r are identifying the partners that made the communication happen; or external transitions with labels of the form (u, r, a) with a an ordinary action, u identifying the process responsible for a and r being an identifier provided by the context.

2.4.1 Basic rules

We first have two basic transitions:

$$\frac{}{M[a.Q + P]_u \xrightarrow{u,r,a} \langle r,a,P\rangle.M[Q]_u}\ \textbf{(com)}$$

$$\frac{}{\langle r,a,P\rangle.M[Q]_u \xleftarrow{u,r,a} M[a.Q + P]_u}\ \textbf{(back)}$$

One observes that **(back)** which is undoing a communication is *also* a communication. It is the exact inverse to **(com)**. To keep the system consistent one is allowed to roll-back if and only if the other former partner in communication is willing to.

We will use sometimes irreversible or unbacktrackable actions, written $\underline{a}$. For such actions communication is simply not memorized, and one uses the simplified **(com)′**:

$$\frac{}{M[\underline{a}.Q + P]_u \xrightarrow{u,r,\underline{a}} M[Q]_u}\ \textbf{(com)}'$$

2.4.2 Contextual rules

We turn to the definition of contextual rules, explaining what becomes of a basic transition when done in a context.

Below, when we no longer want to distinguish between foward communication and backward communication, we write $\xleftrightarrow{u,r,a}$ meaning either $\xrightarrow{u,r,a}$ or $\xleftarrow{u,r,a}$.

$$\frac{P \xleftrightarrow{u,r,a} P'}{\langle\rangle[P]_u \xleftrightarrow{u,r,a} \langle\rangle[P']_u}\ (\xi)$$

$$\textbf{(par)}\ \frac{P \xleftrightarrow{u,r,a} P'}{(P \mid Q) \xleftrightarrow{u,r,a} (P' \mid Q)} \qquad \frac{P \xleftrightarrow{u,r,a} P' \qquad a \neq c, \bar{c}}{(\nu c)P \xleftrightarrow{u,r,a} (\nu c)P'}\ \textbf{(res)}$$

$$\textbf{(syn)}\ \frac{P_1 \xleftrightarrow{u,r,a} P_1' \qquad P_2 \xleftrightarrow{r,u,\bar{a}} P_2'}{(P_1 \mid P_2) \xleftrightarrow{u,r,\tau} (P_1' \mid P_2')} \qquad \frac{P \equiv P' \quad P' \xleftrightarrow{u,r,a} Q' \quad Q' \equiv Q}{P \xleftrightarrow{u,r,a} Q}\ (\equiv)$$

There is nothing remarkable in these rules except that the (ξ) rule is requiring that the memory stack guarding a given thread be *empty* before any action involving its sub-threads can take place. This rule, used with the **(share)** congruence rule, is the core of CCS-R system (see example 1 below).

We will write $P \overset{*}{\longleftrightarrow} P'$, meaning there is a succession of transitions, including possibly zero, starting from P and leading to P'.

2.5 Causality and Backtrack

Labels in our transition system are not unlike "proof terms" used to label CCS transitions in [5,4]. The original motivation for introducing these proof terms was to have enough information to analyze the causal dependencies between actions. Not every action done after an action a is actually depending on a. For example, both $a|b$ and $a.b$ can produce the computation trace $a \cdot b$, but only in the latter case is b depending on a, in the former a and b are concurrent.

A proper backtrack mechanism has to perform some implicit causality analysis, or else one could backtrack on some action a without backtracking first on a b that a has made to happen: a disaster ! So it is perhaps unsurprising that there is a family resemblance with causality-oriented systems.

2.6 Sharing and Linking

From now on, we will write simply $[P]_u$ instead of $\langle\rangle[P]_u$ and use M_u to denote the memory stack of the process identified by the identifier u.

*Example 1: the need for (**share**).*

Consider the system:

$$\mathcal{S} = [x.([y.0 + R]_{11} \mid [\bar{y}.0]_{12}) + Q\,]_1$$

and suppose for a moment there isn't any restriction on the (ξ) rule, then the following sequence of transitions becomes possible:

$$\begin{aligned}
\mathcal{S} &\xrightarrow{1,r,x} \quad \mathcal{S}_1 = \langle r,x,Q\rangle[\,[y.0 + R]_{11} \mid [\bar{y}.0]_{12}]_1 \\
&\xrightarrow{11,12,\tau} \quad \mathcal{S}_2 = \langle r,x,Q\rangle[\langle 12,y,R\rangle[0]_{11} \mid \langle 11,\bar{y},0\rangle[0]_{12}\,]_1 \\
&\xleftarrow{1,r,x} \quad \mathcal{S}_1' = [x.(\langle 12,y,R\rangle[0]_{11} \mid \langle 11,\bar{y},0\rangle[0]_{12}) + Q\,]_1
\end{aligned}$$

The reduction inside $[\,]_1$ resulting in $\mathcal{S}_2$ is a problem, because after this one could backtrack on M_1 and produce a process corresponding to $\phi(\mathcal{S}_1') = x.(0 \mid 0) + Q$ in CCS, which is unreachable from $\phi(\mathcal{S})$. The point of the restriction of the (ξ) rule is to forbid the direct transition from $\mathcal{S}_1$ to $\mathcal{S}_2$. Apart from an immediate backtrack, only **(share)** can be applied to $\mathcal{S}_1$:

$$\begin{aligned}
\mathcal{S}_1 \equiv \quad & \langle\rangle[\langle r,x,Q\rangle[y.0 + R]_{11} \mid \langle r,x,Q\rangle[\bar{y}.0]_{12}]_1 \\
\xrightarrow{11,12,\tau} \quad & \langle\rangle[\langle 12,y,R\rangle.\langle r,x,Q\rangle[0]_{11} \mid \langle 11,\bar{y},0\rangle.\langle r,x,Q\rangle[0]_{12}]_1
\end{aligned}$$

but then memories are set correctly and the obtained process has to first to backtrack on the internal communication on y before undoing the communication on x.

*Example 2: the rôle of **(link)**.*

Consider now the system $[K]_1$ with $K := x.K + Q$; if one could freely unfold a constant without keeping track of this in the associated memory, then we would get this:

$$[K]_1 \xrightarrow{1,r_1,x} \langle r_1,x,Q\rangle[K]_1 \qquad \xrightarrow{1,r_2,x} \langle r_2,x,Q\rangle\langle r_1,x,Q\rangle[K]_1$$
$$\xleftarrow{1,r_2,x} \langle r_1,x,Q\rangle[x.K+Q]_1 \xleftarrow{1,r_1,x} \langle\rangle[x.(x.K+Q)+Q]_1$$

The resulting process is structurally equivalent to the original one, but one wants to force the re-folding of unfolded definitions when rolling back and this is why one records unfoldings in memory. In our example, we get:

$$\begin{aligned}
[K]_1 &\equiv \langle def{:}K(1)\rangle[x.K+Q]_1\\
&\xrightarrow{1,r_1,x} \langle r_1,x,Q\rangle.\langle def{:}K(1)\rangle[K]_1\\
&\equiv \langle def{:}K(1)\rangle.\langle r_1,x,Q\rangle.\langle def{:}K(1)\rangle[x.K+Q]_1\\
&\xrightarrow{1,r_2,x} \langle r_2,x,Q\rangle.\langle def{:}K(1)\rangle.\langle r_1,x,Q\rangle.\langle def{:}K(1)\rangle[K]_1\\
&\xleftarrow{1,r_2,x} \langle def{:}K(1)\rangle.\langle r_1,x,Q\rangle.\langle def{:}K(1)\rangle[x.K+Q]_1\\
&\equiv \langle r_1,x,Q\rangle.\langle def{:}K(1)\rangle[K]_1\\
&\xleftarrow{1,r_1,x} \langle def{:}K(1)\rangle[x.K+Q]_1\\
&\equiv \langle\rangle[K]_1
\end{aligned}$$

and **(link)** is recovering the exact original process.

2.7 Soundness

Now that the system has a clearcut definition and some examples, it is time to prove that it actually works as expected.

Lemma 2.1 (Reversibility)

$$\forall P \in \textit{CCS-R},\ P \xrightarrow{u,r,a} P' \iff P' \xleftarrow{u,r,a} P$$

Both implications are shown by induction on the LTS structure. From this one sees directly that: $P \xleftrightarrow{*} P'$ iff $P' \xleftrightarrow{*} P$, in other words accessibility is a symmetric relation, which is a rigorous way of saying that our transition system is reversible.

Lemma 2.2 (Soundness)

$$\forall P \in \textit{CCS-R},\ P \xrightarrow{u,r,a} P' \Rightarrow \phi(P) \xrightarrow{a} \phi(P')$$

where $\xrightarrow{a}$ is a CCS transition.

Again the proof is by induction on the LTS.

Lemma 2.3 (Completeness) $\forall P \in \mathit{CCS}\text{-}R$,

$$\phi(P) \xrightarrow{a} Q \Rightarrow \exists u, r, Q' : P \xrightarrow{u,r,a} Q' \wedge \phi(Q') = Q$$

The proof is by induction on P.

Theorem 2.4 (Equivalence)

$$\forall P \in \mathit{CCS}\text{-}R_i,\ P \xleftrightarrow{*} P' \iff \phi(P) \xrightarrow{*} \phi(P')$$

Take note that P has to be in CCS-R$_i$ for this theorem to be true, else some memory might be hidden in P. An evident example is $P = \langle r,a,Q\rangle[0]_1$ which can roll-back to $\langle\rangle[a.0 + Q]_1$ while $\phi(P) = 0$ has, by definition, no transition in CCS.

All the proofs above are routine; the theorem follows immediately from the lemmas.

3 Formalizing biological systems

Let us close the process algebra chapter and turn to the biological question. We first explain our mental picture of biological interactions and in so doing we perform various modifications on our basic language to adapt it to the specific task of representing such interactions.

3.1 Elasticity and Plasticity

In a solution, several complexes may form. Some of them will be stable, others will not, but indeed proteins do not have the knowledge of what is stable and what is not. So in our mental model, proteins always try to bind with any complementary interface present in the solution, while nature tries to bring back reactants in their original states. This principle of protein *elasticity* is illustrated in Fig 3. In the absence of any constraint on the spatial configuration of a protein, such as a complexation or an activation (see below), it will always fold back to its original configuration. This fits perfectly with our backtrack mechanism.

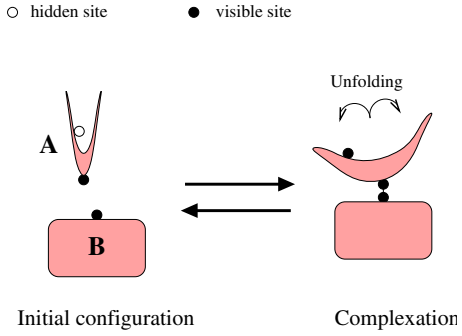

Fig. 3. *Protein elasticity*

That said, complexations are not the only way that a protein can change its interaction capabilities. It can also be phosphorylated or activated otherwise.[3] So there also is a certain amount of *plasticity* going on. Fig. 4 gives an illustration of this: $B^\star$ represents a phosphorylated state of B. Since activation has changed the natural configuration of B, it won't come back to its original form after decomplexation. Activation is a permanent change of the natural configuration of a protein and it therefore seems natural to use irreversible or unbacktrackable actions to model it.

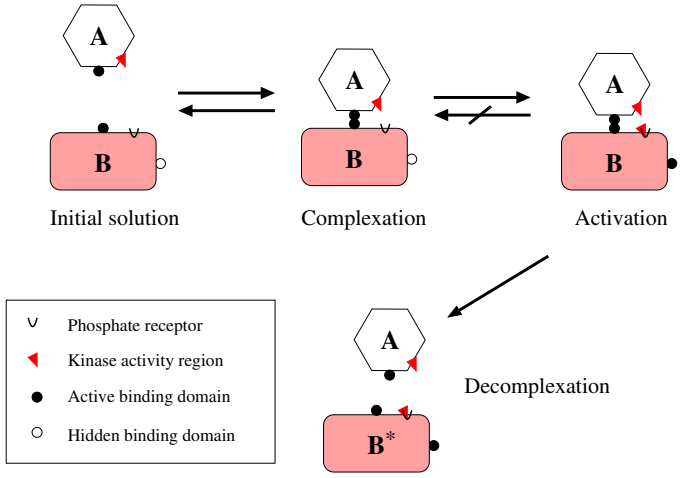

Fig. 4. *Protein plasticity*

3.2 Adapting CCS-R

All this requires a few adaptations to the system.

First, the simple CCS duality between names a and co-names $\bar{a}$ has to be generalized to a binary complementation relation written $\mathcal{C}$ between binding sites, two binding sites x and x' being able to connect to each other if $x\,\mathcal{C}\,x'$.

Second, we observe that sometimes biological protein-protein interactions require a *concurrent* connexion on several sites, before allowing another reactant to connect to the complex, as shown in Fig. 5.

To write C as $(l_1|l_2|l_3).l_4$ seems the obvious thing to do, but such an expression is not part of the official syntax. Usually this kind of extension of the process syntax is not considered, since one can always rely on the expansion law to cast the process C in ordinary form. In our example, C would be expanded as $l_1.l_2.l_3.l_4+l_1.l_3.l_2.l_4+\ldots$, unfortunately, this introduces a fake causal dependency.[4,5]

[3] During a phosphorylation or de-phosphorylation, a phosphate is exchanged between the reactants resulting in covalent binding or unbinding. This may change the protein state permanently.

[4] A similar argument is made in [2].

[5] Another possibility is to define C as $(\nu t_i)(l_1.t_1|l_2.t_2|l_3.t_3|t_1.t_2.t_3.l_4)$, but the t_is have no biological contents.

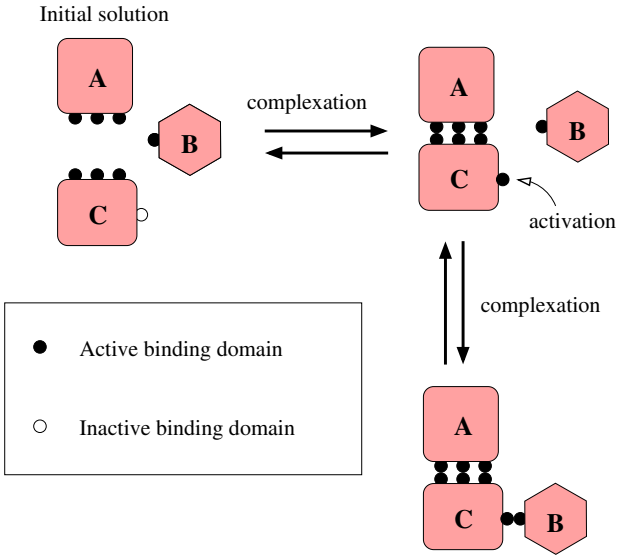

Fig. 5. *A binds B then B binds C.*

One can see this even with a much simpler example:

$$\begin{aligned}[a.0]_1 \mid [b.0]_2 &\xrightarrow{1,r,a} \langle r,a,0\rangle[0]_1 \mid [b.0]_2\\ &\xrightarrow{2,s,b} \langle r,a,0\rangle[0]_1 \mid \langle s,b,0\rangle[0]_2\\ &\xleftarrow{1,r,a} [a.0]_1 \mid \langle s,b,0\rangle[0]_2\end{aligned}$$

but after expanding $[a]_1 \mid [b]_2$ as $a.b + b.a$ the trace above is not valid anymore. In this specific sense, the expansion law is wrong when it comes to backtracking.

3.3 Multi-actions

So we do this differently by letting processes be guarded by *multisets* of actions, or multi-actions. The ordinary case is when these multisets are just singletons. Multi-actions can be performed *and* backtracked in any order. Multi-actions are given by: [6]

$$\begin{array}{lll}\text{Multi-actions:} & \mu ::= \epsilon \mid (\mu|a) & \\ \text{Tagged actions:} & \gamma ::= \epsilon \mid ((r,a)|\gamma) & \\ \text{Framed-memories:} & M ::= \ll_{\gamma;P}\gg.M & \text{Opened memory}\\ & \quad \mid \langle\gamma;P\rangle.M & \text{Closed memory}\end{array}$$

where ϵ stands for the empty multiset, r is an identifier and γ a multiset of pairs (r,a) with a an action (not a multi-action !). One sees that memories can now be *opened* or *closed*. Actions belonging to a same multi-action will be stored in

[6] Our multi-actions have nothing to do with the patterns of Join-calculus [6,7], since they are synchronized independently.

the same frame of opened memory, at the top of the stack. Take note that since multi-actions are multisets, actions a within a multi-action γ freely commute one with another.

The **(com)** and **(back)** rules have to be modified accordingly:

$$\frac{}{\ll\gamma;P\gg.M[(\mu|a).Q]_u \xrightarrow{u,r,a} \ll\gamma|(r,a);P\gg.M[\mu.Q]_u} \ (\mu\textbf{-com})$$

$$\frac{}{\ll\gamma|(r,a);P\gg.M[\mu.Q]_u \xleftarrow{u,r,a} \ll\gamma;P\gg.M[(\mu|a).Q]_u} \ (\mu\textbf{-back})$$

Frame opening and closing is dealt with by additional structural equations:

$$M[\mu.P+Q]_u \equiv \ll\epsilon;Q\gg.M[\mu.P]_u \ \textbf{(open)}$$
$$\ll\gamma;Q\gg.M[\epsilon.P]_u \equiv \langle\gamma;Q\rangle.M[P]_u \qquad \textbf{(close)}$$

with the condition that M is itself closed in the **(open)** equation.

As said, CCS-R can be embedded in this extension, which we call CCS-Rμ, as the fragment where γs and μs are always singletons. Again the new system is only a decoration of CCS.

3.4 Back to the Example

Now we have all the material to give a satisfying representation of the situation summarized in Fig. 5. Define:

$$A := (l_1|l_2|l_3).0 \qquad B := l_b.0 \qquad C := (l'_1|l'_2|l'_3).l_4.0$$

with for all i, $l_i \,\mathcal{C}\, l'_i$ and $l_b \,\mathcal{C}\, l_4$. Now we get the following computation trace:

$$\begin{aligned}
\mathcal{S} = \quad & [A]_1 \mid [B]_2 \mid [C]_3 \\
\xrightarrow{1,3,\tau} \ & \ll(3,l_1);0\gg[\{l_2,l_3\}.0]_1 \mid [B]_2 \mid \ll(1,l'_1);0\gg[\{l'_2,l'_3\}.l_4.0]_3 \\
\xrightarrow{1,3,\tau} \ & \ll(3,l_1)|(3,l_3);0\gg[\{l_2\}.0]_1 \mid [B]_2 \mid \ll(1,l'_1)|(1,l'_3);0\gg[\{l'_2\}.l_4.0]_3 \\
\xrightarrow{1,3,\tau} \ & \langle(3,l_1)|(3,l_3)|(3,l_2);0\rangle[0]_1 \mid [B]_2 \mid \langle(1,l'_1)|(1,l'_3)|(1,l'_2);0\rangle[l_4.0]_3
\end{aligned}$$

and as expected, the connexion of l_i to l'_i can be performed *and* backtracked in any order. Specifically any tagged action in $(1,l'_1)|(1,l'_3)|(1,l'_2)$ can trigger a backward transition. The final complexation is obtained with:

$$\xrightarrow{2,3,\tau} \langle(3,l_1)|(3,l_3)|(3,l_2);0\rangle[0]_1 \mid \langle(3,l_b);0\rangle[0]_2 \mid \langle(2,l_4);0\rangle.\langle(1,l'_1)|(1,l'_3)|(1,l'_2);0\rangle[0]_3$$

4 CCS-R expressivity

More elaborate biological situations can be represented. Here we formalize a system where the transcription of a protein is controlled by a competition between different reactants. Backtracking and multi-actions are both needed.

Before the gene transcription starts, a complex protein machinery, denoted by A, has to bind upstream of the gene on the DNA strand. In our example, there are two inhibitors: $I2$ which is competing with A to bind to the same site on DNA, and $I1$ which prevents the DNA strand from folding properly. The situation is summarized in Fig. 6. Here is the representation in CCS-Rμ:

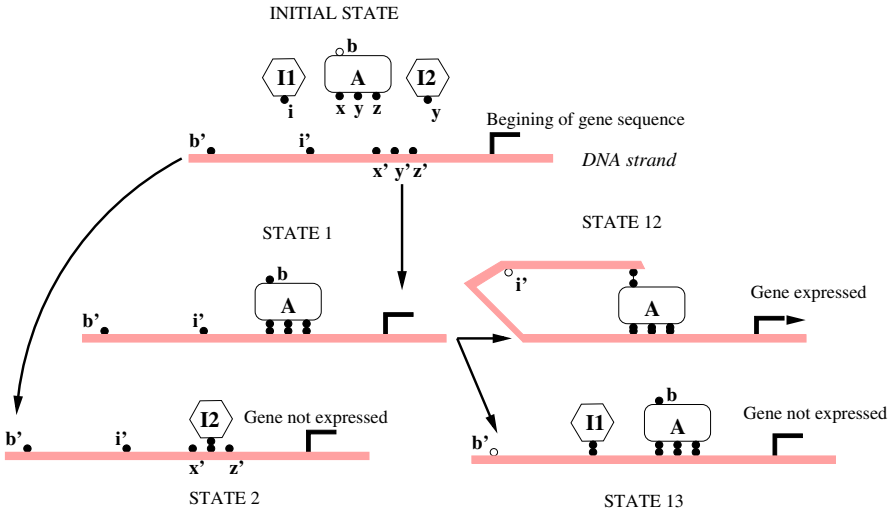

Fig. 6. *Regulation of a gene expression*

$$
\begin{aligned}
A &:= (x|y|z).b.\mathrm{EXP} \\
\mathrm{DNA} &:= [(x'|y'|z').0]_{11} \mid [(b'.\mathrm{EXP}' + i'.0)]_{12} \\
I1 &:= i.0 \\
I2 &:= y.0
\end{aligned}
$$

where EXP and EXP$'$ are constants handling the gene expression phase, and the codomain relation $\mathcal{C}$ is such that (x, x'), (y, y'), etc. are in $\mathcal{C}$. Supposing the initial solution is:

$$\mathcal{S} = [\mathrm{DNA}]_1 \mid [A]_2 \mid [I1]_3 \mid [I2]_4,$$

we have here a possible computation trace:

$$\begin{aligned}
\mathcal{S} \xrightarrow{11,2,\tau} & \ll_{(2,x');0}\gg[(y'|z').0]_{11} \mid [b'.EXP' + i'.0]_{12} \\
& \mid \ll_{(1,x);0}\gg[(y|z).b.EXP]_2 \\
& \mid [I1]_3 \\
& \mid [I2]_4 \\
\\
\xrightarrow{11,4,\tau} & \ll_{(2,x')|(4,y');0}\gg[z'.0]_{11} \mid [b'.EXP' + i'.0]_{12} \\
& \mid \ll_{(1,x);0}\gg[(y|z).b.EXP]_2 \\
& \mid [I1]_3 \\
& \mid \langle_{(1,y);0}\rangle[0]_4 \\
\\
\xrightarrow{11,2,\tau} & \langle_{(2,x')|(4,y')|(2,z');0}\rangle[0]_{11} \mid [b'.EXP' + i'.0]_{12} \\
& \mid \ll_{(1,x)|(1,z);0}\gg[y.b.EXP]_2 \\
& \mid [I1]_3 \\
& \mid \langle_{(1,y);0}\rangle[0]_4
\end{aligned}$$

For the sake of clarity, linkings were not traced in this run, neither was the $[\,]_1$ bracing DNA repeated along the run as it should.

At this stage A and $I2$ are both bound to DNA and one of them has to disconnect (recall the example in the introduction). Suppose that $I2$ let go first, then computation may proceed in this way:

$$\begin{aligned}
\xleftarrow{11,4,\tau} & \ll_{(2,x')|(2,z');0}\gg[y'.0]_{11} \mid [b'.EXP' + i'.0]_{12} \\
& \mid \ll_{(1,x)|(1,z);0}\gg[y.b.EXP]_2 \\
& \mid [I1]_3 \\
& \mid \langle\rangle[y.0]_4 \\
\xrightarrow{11,2,\tau} & \langle_{(2,x')|(2,y')|(2,z');0}\rangle[0]_{11} \mid [b'.EXP' + i'.0]_{12} \\
& \mid \langle_{(1,x)|(1,z)|(1,y);0}\rangle[b.EXP]_2 \\
& \mid [I1]_3 \\
& \mid \langle\rangle[y.0]_4
\end{aligned}$$

Here, transcription may start with $\stackrel{12,2,\tau}{\longrightarrow}$ or be inhibited the other way:

$$\begin{aligned}\stackrel{12,3,\tau}{\longrightarrow}\ & \langle(2,x')|(2,y')|(2,z');0\rangle[0]_{11} \mid \langle(3,i');b'.EXP\rangle[0]_{12} \\ & \mid \langle(1,x)|(1,z)|(1,y);0\rangle[b.EXP]_2 \\ & \mid \langle\{(1,i)\},0\rangle[0]_3 \\ & \mid [y.0]_4\end{aligned}$$

Each transition corresponds to a biological event and in particular, backward steps are faithfully representing unbinding.

5 Discussion and future work

So far, we have considered systems in which one could assume that reactants were free to "move" in the solution to connect to any complementary interface. Speculating about what can be done when this perfect mix assumption doesn't hold is now the topic. In this case we have a problem best explained with an example. We choose the beginning of the RTK-MAPK cascade already modeled in [17] and illustrated in Fig. 7.

The graphic representation of the system indicates that the EGF receptor EGFR cannot activate itself though it has both a kinase activity region KAR and a phosphate receptor site k. If we take as initial solution:

$$\mathcal{S} = [\mathrm{EGF}]_1 \mid [\mathrm{EGF}]_2 \mid [\mathrm{EGFR}]_3 \mid [\mathrm{EGFR}]_4 \mid [\mathrm{SHC}]_5 \mid [\mathrm{SHC}]_6$$

together with the following process definitions:

$$\begin{aligned}\mathrm{EGF} &:= a.0 \mid b'.0 \\ \mathrm{EGFR} &:= \mathrm{KAR} \mid b'.\underline{k}.\mathrm{EGFR}^\star \\ \mathrm{EGFR}^\star &:= s.0 \\ \mathrm{KAR} &:= \bar{\underline{k}}.\mathrm{KAR} \\ \mathrm{SHC} &:= s'.d.0\end{aligned}$$

and the following list of complementary sites $\mathcal{C} = \{(a,a),(b,b'),(s,s'),(\underline{k},\bar{\underline{k}})\}$, we see that the normal CCS-R representation is wrong here, since it allows self-phosphorylation

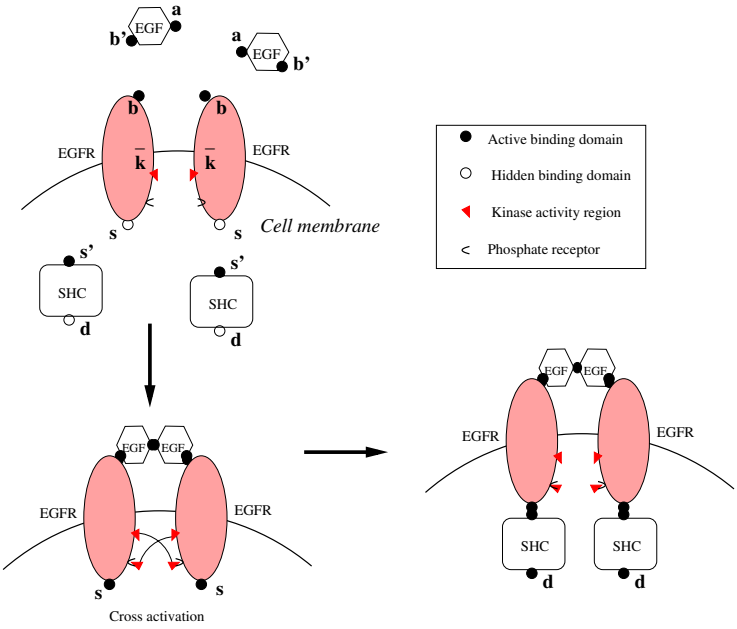

Fig. 7. *RTK/MAPK cascade: initial solution*

of EGFR:

$$\mathcal{S} \xrightarrow{11,21,\tau} \langle(21,a);0\rangle[0]_{11} \mid [b.0]_{12} \mid \langle(11,a);0\rangle[0]_{21} \mid [b.0]_{22} \mid [\mathrm{EGFR}]_3 \mid [\mathrm{EGFR}]_4 \mid [\mathrm{SHC}]_5 \mid [\mathrm{SHC}]_6$$

$$\xrightarrow{12,32,\tau} \langle(21,a);0\rangle[0]_{11} \mid \langle(32,b);0\rangle[0]_{12} \mid \langle(11,a);0\rangle[0]_{21} \mid [b.0]_{22} \mid [\mathrm{KAR}]_{31} \mid \langle(12,b');0\rangle[\underline{k}.\mathrm{EGFR}^\star]_{32} \mid [\mathrm{EGFR}]_4 \mid [\mathrm{SHC}]_5 \mid [\mathrm{SHC}]_6$$

$$\xrightarrow{31,32,\tau} \langle(21,a);0\rangle[0]_{11} \mid \langle(32,b);0\rangle[0]_{12} \mid \langle(11,a);0\rangle[0]_{21} \mid [b.0]_{22} \mid [\mathrm{KAR}]_{31} \mid \langle(12,b');0\rangle[\mathrm{EGFR}^\star]_{32} \mid [\mathrm{EGFR}]_4 \mid [\mathrm{SHC}]_5 \mid [\mathrm{SHC}]_6$$

Of course we can modify our encoding of the system, by distinguishing two different kinds of EGFR, the left one and the right one. This prevents self-activation but to the price of breaking the symmetry between the two receptors. Or else, we can easily code this by falling back to a value-passing mechanism. Should one give in to such an extension of the basic language ? We don't have a good answer at the

moment.

6 Conclusion

We have constructed an extension of CCS that incorporates backtracking with a relatively low syntactic cost. Some further amendments motivated by the desire to represent biological systems in the neatest way have been introduced. In particular, we had to remedy the "fake dependencies" problem since we have seen that expansion laws are wrong in a reversible system. This has led us to introduce multi-actions. (One can probably go further and give a complete treatment of the full sequential composition $P;Q$ of which multi-actions are a particular instance). We also discovered an interesting way of representing state-changes through unbacktrackable actions.

However, CCS-R has shortcomings, as explained in the concluding section, and cannot properly express situations where active complementary sites cannot always bind one another. It is not clear at the moment if there is a clean answer to this problem.

On the foundational level, a question remains to be answered, namely whether reversibility can be encoded in a principled way in π-calculus. Preliminary investigations suggest that the answer is positive but this remains to be verified. One interest of a nice and definite answer here would be to give a measure (an upper bound really) on how much modeling effort is discounted when working in a system where backtracking is built-in and not something one has to code for.

A variant of CCS comparable to CCS-R was never considered before as far as we know. This is perhaps for want of any motivation. Trying to fit some biological trait in a formal picture is indeed a great source of inspiration. Once this is done one may perhaps forget the original motivation, at least for a moment and look for other kinds of applications. In our own case, the representation of decentralized transactional mechanisms comes to mind, but work is needed here to see if something interesting can be done and we leave this for further investigation.

References

[1] C. Baldi, Pierpaolo Degano, and Corrado Priami. Causal π-calculus for biochemical modeling. In *Proceedings of the AI*IA Workshop on BioInformatics 2002*, pages 69–72, 2002.

[2] Michele Boreale and Davide Sangiorgi. A fully abstract semantics for causality in the π-calculus. *Acta Inf.*, 35:353–400, 1998.

[3] Vincent Danos and Cosimo Laneve. Graphs for formal molecular biology. In *Proceedings of the First International Workshop on Computational Methods in Systems Biology (CMSB'03, Rovereto, Italy)*, volume 2602 of *LNCS*, pages 34–46. Springer-Verlag, 2003.

[4] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. A partial ordering semantics for CCS. *Theoretical Computer Science*, 75:223–262, 1990.

[5] Pierpaolo Degano and Corrado Priami. Proved trees. In *Automata, Languages and Programming*, volume 623 of *LNCS*, pages 629–640, 1992.

[6] Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *23rd ACM Symposium on Principles of Programming Languages (POPL'96)*, 1996.

[7] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of CONCUR'96*, 1996.

[8] Ronojoy Ghosh and Claire J. Tomlin. Lateral inhibition through delta-notch signaling: A piecewise affine hybrid model? In *Proceedings of HSCC 2001*, volume 2034 of *LNCS*, pages 232–246, 2001.

[9] R. Hofestädt and S. Thelen. Quantitative modeling of biochemical networks. *In Silico Biology*, 1, 1998.

[10] Trey Ideker, Vesteinn Thorsson, Jeffrey A. Ranish, Rowan Christmas, Jeremy Buhler, Jimmy K. Eng, Roger Bumgarner, David R. Goodlett, Ruedi Aebersold, and Leroy Hood. Integrated genomic and proteomic analyses of a systematically perturbed metabolic pathway. *Science*, 292:929–934, May 2001.

[11] Naaman Kam, Irun R. Cohen, and David Harel. The immune system as a reactive system: modeling T-cell activation with statecharts. In *Proceedings of the IEEE Symposium on Human-centric Computing*, pages 15–22, September 2001.

[12] KEGG. Kyoto Encyclopedia of Genes and Genomes, Bioformatics Center, Institute for Chemical Research, Kyoto University.

[13] H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid Petri Net representation of gene regulatory networks. In *Proceedings of the Pacific Symposium on Biocomputing (2000)*, pages 338–349, 2000.

[14] Robin Milner. *Communication and Concurrency*. International Series on Computer Science. Prentice Hall, 1989.

[15] Corrado Priami, Aviv Regev, William Silverman, and Ehud Shapiro. Application of stochastic name-passing calculus to representation and simulation of molecular processes. *Infomation Processing Letters*, 80(1):25–31, 2001.

[16] Aviv Regev and Ehud Shapiro. Cells as computation. *Nature*, 419, September 2002.

[17] Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 6, pages 459–470. World Scientific Press, 2001.

[18] Birgit Schoeberl, Claudia Eichler-Jonsson, Ernst-Dieter Gilles, and Gertraud Müller. Computational modeling of the dynamics of the map kinase cascade activated by surface and internalized EGF receptors. *Nature Biotechnology*, 20:370–375, 2002.