Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 325 (2004) 69–110
Theoretical Computer Science
www.elsevier.com/locate/tcs

Formal molecular biology

Vincent Danos[a,*], Cosimo Laneve[b]

[a] *Équipe PPS, CNRS & Université Paris 7, Paris, Cedex 75251, France*
[b] *Department of Computer Science, University of Bologna, France*

Abstract

A language of formal proteins, *the κ-calculus*, is introduced. Interactions are modeled at the domain level, bonds are represented by means of shared names, and reactions are required to satisfy a causality requirement of *monotonicity*.

An example of a simplified signalling pathway is introduced to illustrate how standard biological events can be expressed in our protein language. A more comprehensive example, the lactose operon, is also developed, bringing some confidence in the formalism considered as a modeling language.

Then a finer-grained concurrent model, the *mκ-calculus*, is considered, where interactions have to be at most binary. We show how to embed the coarser-grained language in the latter, a property which we call *self-assembly*.

Finally we show how the finer-grained language can itself be encoded in π-calculus, a standard foundational language for concurrency theory.

1. Introduction

Following independent proposals from Fontana [11] and Regev [25], it is becoming commonplace to think of formalisms derived from process algebras and concurrency as being potentially useful in the formal layout and analysis of biological networks at the molecular level. We will first restate the goals of this relatively recent movement of ideas before explaining the contribution of this particular paper.

The cell is a billion moving pieces implementing life. Sugar is collected, processed and used as a power supply to gather information. The better the cell is fed, the better it computes. Signals are detected, collected and compared and some decisions are taken. The better the cell computes, the better it feeds on the environment. Life needs sugar

* Corresponding author.
E-mail address: danos@logique.jussieu.fr (V. Danos).

0304-3975/$ - see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.tcs.2004.03.065

to process information and also direly needs information to process sugar. One could be left wondering if this is perhaps the meaning of life, but meanwhile there must be a programming lesson to be understood here.

To begin with, computation in a cell is concurrent and asynchronous. Synchronization when it is needed—for instance to detect the presence of significant amounts of two signals at the same time—has to be implemented. Second, the systems semantics depends on probabilistic response and yet often remains deterministic at the macroscopic level. Values manipulated are mostly continuous, yet discrete states and choices can be implemented at various levels; behaviors are obtained through heavy use of feedback mechanisms. Delays are involved, some computation steps are partly reversible, rates of reactions are influenced by global parameters and perhaps as a consequence, local environments can be created by means of compartments.

Bio-computing is in radical departure from ordinary computational models, where a sequence of actions is chosen once and for all, data is discrete, control is either centralized or at most coarsely distributed, following design principles that seek to optimize efficiency at fairly simple and well-defined tasks.

Considering the recent breakthroughs in experimental biology, we may be for the first time in position to understand *what computational models are embedded in the cell* and to develop a symbolic biology that would be at the same time a manageable theoretical object and a plausible idealization.

Some of this excitement has already transpired in the domain of Concurrency. New process algebras directed at biological systems are now mushrooming, each meant to treat one aspect of the specificities of bio-computing. One has reversible CCS [6] giving means to directly express reversibility, stochastic π-calculus [22,23] equipped with a quantitative contextual semantics and therefore giving access to simulations, bio-ambients and membrane-calculi [3,24] for dealing with the dynamics of various cellular compartments. We ourselves proposed a first version of the κ-calculus with the specific purpose of representing protein interactions [8]. The present paper works a refined version of this same language.

1.1. A calculus for proteins

Our language idealizes protein–protein interactions, essentially as a particular restricted kind of graph-rewriting operating on graph-with-sites not unlike Lafont's interaction nets [19]. Bindings are explicit: a formal protein is a node with a fixed number of sites, a complex is a connected graph built over such nodes.

Biological reactions are modeled by two kinds of rewriting rules: monotonic and antimonotonic. The former kind represents complexations, that is to say reactions where low energy bonds are formed between various compounds. The latter kind is symmetric to the first and represents decomplexation. From this respect, reaction formats are more restrictive than they were in the preceding version of the language [8]. Yet, they are also more expressive, in that they allow non-linear reactions and thus make it possible to represent the important reactions of synthesis and degradation. This is a significant rise in expressive power at a low syntactic price.

We illustrate this gain in expressivity with a typical signal transduction pathway. A more comprehensive formalization effort was done previously. Using a simplified and more abstract relative of κ-calculus, the vast network of reactions controlling the mammalian cell cycle was formalized [5] after Kohn's compilation [18]. Take note that, in signal transduction systems, synthesis can be taken as an output and does not need to be modeled in itself. In the cell cycle things are different. Synthesis plays a major role because the chief regulators of the cell cycle, called the cyclins, are influencing their own synthesis through a complicated cascade of interactions. One really has to describe it inside the model. In principle, this formalization could be recast in our present language, provided one has enough detail about domain-level interactions. We choose here, as a second example, a simpler and well documented system, namely the lactose operon. While remaining tractable the set of reactions involved offers a selection of most of the events one finds in the bigger mammalian cell cycle control.

Both examples serve well as a practical proof of the expressivity of our new version of κ-calculus and the second puts to use its additional expressivity power. Another difference with the previous version is that we work now with an algebraic notation instead of keeping with a graph-rewriting presentation. This process algebraic notation is closer to a multiset-based calculus which we proposed earlier [7] and allows for a more flexible and precise syntax for reactions.

As for any process algebra, once the basic reactions are in place, one can derive the behavior of any system by means of contextual rules. The possibility of applying pattern-based basic reactions in different contexts brings an element of prediction in the language. In this respect κ-calculus is not equivalent to a flat and reaction-centric view of biological systems. Bringing a notion of contextual quantitative operational semantics à la Gillespie [13] as Priami did for stochastic π-calculus [22,23] could make this even more evident since unexpected evolutions of the system would be observed. But for now we do not have such a quantitative semantics and this interesting issue remains to be explored.

1.2. Self-assembly

As pleasingly simple and close to biological interactions as our language may be, even the restricted reactions it considers cannot plausibly be taken as atomic events. Be that in biology, or in any other decentralized computational scenario for that matter, non-local graph-rewriting takes time and more accurately it takes consensus. To implement this consensus is a problem, which after Klavins [17], we name the *self-assembly* problem. In our specific case, the informal question becomes whether given a higher level description in κ-calculus, one can synthesize processes, one for each of the interacting proteins, so that in a purely decentralized way and with binary synchronization as the only means of communication, the proteins are going to behave according to the original higher level description.

The second contribution of this paper is to make rigorous sense of the self-assembly question and solve it to the positive for our language. Though the π-calculus is a process language that is well suited to a formalization of self-assembly, we introduce an intermediate language, the $m\kappa$-calculus, that allows for a more readable formulation and

solution of our problem. The finer-grained view of protein interaction that $m\kappa$-calculus introduces is interesting in its own right and one might argue that it is biologically more plausible. Basic entities are called agents, they have a state, may create names and communicate only by binary interactions. To encode a given higher level reaction, one uses a family of binary interactions indexed by the edges of a directed acyclic graph spanning all the reactants. Such a graph is guaranteed to exist by the monotonicity condition. Correctness is obtained through the definition of a simulation which follows this geometric construction. Again, the new rule format compares well with the previous format [8] in terms of the simplicity of the simulation.

Now $m\kappa$-calculus is not far from a graphic notation for π-calculus and we provide an embedding witnessing this in the concluding section. By composing the two encodings, one gets a distributed and non-deterministic implementation of κ-calculus into any current implementation of π-calculus, for instance Nomadic Pict or JoCaml [28,10]. So it is not only that the model we propose is supported by a precise notation and has good descriptive capabilities, but it is also reducible to a protein-centric and purely local language of interactions, such as the π-calculus. These are reference properties against which further models should be evaluated.

1.3. Structure of the paper

The opening section is an informal presentation of our language, and the next section defines the calculus properly and zooms in on the particular format of reactions one is interested in. The next section develops a small example of a signal transduction pathway that illustrates the syntax, and the bigger and well-known example of the lactose operon. Then the paper turns to the matter of self-assembly. A section is devoted to the presentation of the lower level language used to state the problem and the next section presents the actual embedding. The last section proposes an embedding of the lower level language into π-calculus.

2. A visual notation for κ

We begin with a pictorial introduction to our formal calculus. This presentation could be made a formal model in its own right but we choose not to do so, since our working notation, presented in the next section, will be different and actually based on π-calculus rather than on graphs.

So what is it that we want to express in our language? The short answer is *the combinatorics of the interaction between proteins*. Proteins are involved in a network of reactions implementing various high-level tasks such as the sugar-chain repleting energy stocks, the detection of external signals (stress, growth, death,…) and the triggering of the appropriate behavioral modifications, the coordination of internal signals controlling the various phases of the cell cycle, and so on. The main purpose of molecular biology is to identify these tasks and relate them to their implementation at the molecular level.

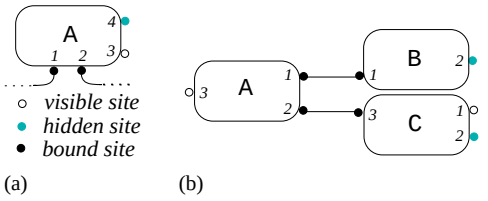


Fig. 1. A protein and a complex.

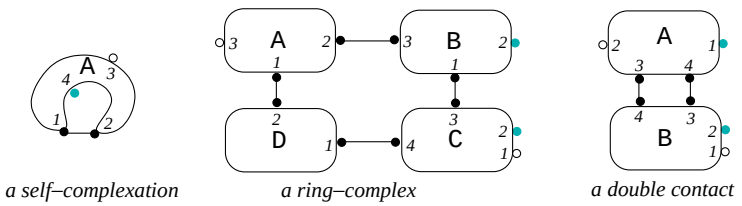


Fig. 2. Complexes.

Going down in the details of protein interaction, one finds sub-components commonly called *domains*, that determine which other proteins they can bind and subsequently interact with. Such interactions may result in changes in the folding of the participating proteins and such changes can sometimes be memorized. There are various ways this is biochemically implemented, the most common being *phosphorylation*. A well-studied protein called P53 is known to have no less than 11 phosphorylation sites and to be able to bind with 12 other proteins to form various binary complexes resulting in an even more daunting combinatorial space [18].

Depending on the way proteins are folded in space, these domains can be active or not, and the behavior of the protein will be different. Therefore not only its free domains but also its global folding determines what a given protein assemblage is capable of.

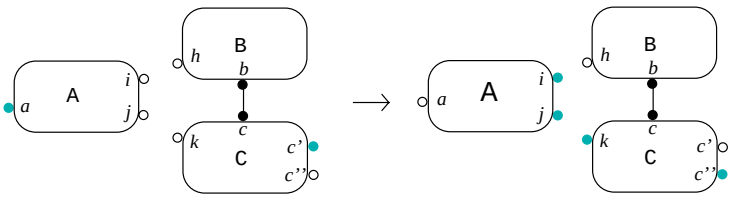
To abstract both over domains and folding states, we use *sites*. These sites may be *bound* or free, and free ones may be *visible* or *hidden*. Thus, in our model, bindings are explicit and internal states are expressed just by saying which free sites are visible or not.

Proteins and complexes. We draw proteins as boxes, with sites being written on the boundary, and identified by distinct natural numbers, 1, 2, 3,..., written within the box. See Fig. 1(a) for a picture of a protein.

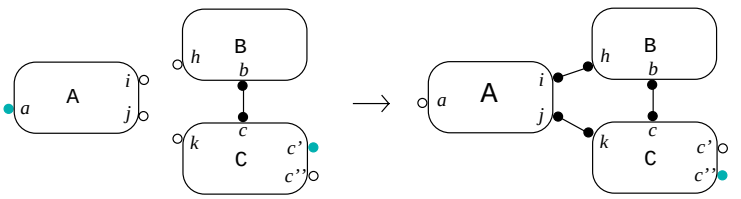
Proteins may be assembled into *protein complexes*, or simply complexes. Complexes are drawn by connecting two-by-two bound sites of proteins, thus building connected graphs such as in Fig. 1(b), which represents a compound made of A, B, and C, where A is connected with B and C. Biologically, a complex is a bundle of proteins connected together by low energy bounds.

Other examples of complexes are shown in Fig. 2.

Biological reactions. Collections of proteins and complexes are called solutions. Solutions evolve by means of reactions, which occur when a sub-solution has a special shape, called a *reactant*. When this happens, the reactants change and yield a new sub-solution. Here are examples of reactions:



activation

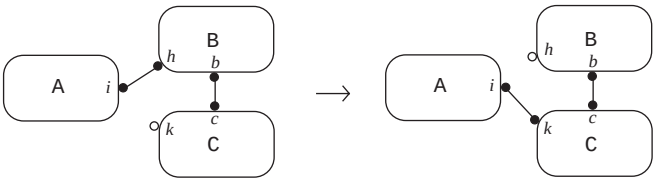


complexation

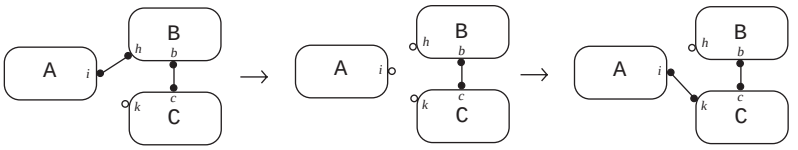
Not every rewriting rule is biologically plausible. We will stipulate in the next section which exact set of reactions we are interested in, but we can already discuss this informally. The complexation reaction above will certainly be in this set, but activation will not. The rationale behind our choice is that activation obviously needs some physical contact to be made to take place. So the reaction above is not an atomic event—we are not told the whole story. By the way, when such activations are found in biological systems, usually only one of the product, called the *substrate* will be modified by the reaction, while the other one called the enzyme, or the kinase, or the catalyst, will be left unchanged. The kinetic analysis of such reactions, embodied in the Michaelis–Menten formula [27], explicitly mentions the intermediate state where the substrate and the catalyst are bound together. All in all, it seems very reasonable not to take that kind of reaction as a primitive.

Roughly, our language will be a language of complexations and decomplexations, where by decomplexation we mean the reaction inverse to complexation when a complex is dissociated into smaller parts. *But* there are two subtle issues here. First, as said in the introduction, it would be too restrictive to only allow linear reactions, that is to say reactions where basic components are preserved. To express the important reactions of synthesis and degradation, we will allow some limited form of duplication and erasing and relax somewhat what would be a too strict preservation principle.

Second, some reactions such as the edge-flipping reaction below, seem to be complexations, but really they are not, because they lack monotonicity.



What do we mean by this? If one looks at the picture, one sees that although both the reactant and product of the reaction are connected, some edge has to be erased in the reactant before one can reach the product. Even if the total number of edges is constant, there must be an intermediate unconnected state of the compounds where an edge is erased, e.g.:



This phenomenon is a consequence of the fact that our model has sites as first-class citizens in the syntax. Sites represent resources for bindings, therefore bindings are constrained by the availability of sites, and this constraint plays an important role in biological causality as we will see in the example pathway.

Speaking about causality, monotonicity (that edges are created and none is erased during a reaction) embodies a causality constraint. For anything to happen things have to be in contact. It is stronger than the virtual connectedness condition that we were imposing in the earlier version of κ-calculus. There we were asking that a temporary super complex could be formed between the reactants. Our new choice makes these temporary complexes explicit, there are either the left hand side for a decomplexation or the right hand side for a complexation. To see that the new condition is strictly stronger it is enough to look at the activation example above. It is virtually connected since the left hand side is connectible through the pairs of visible sites $(A,i;B,h)$ and $(A,j;C,k)$, but as said, it does not satisfy our new requirement.

This new condition is also natural when it comes to the micro-implementation of κ-calculus in $m\kappa$-calculus, in that the complex will be actually walked upon by binary interactions to either check for its existence (case of a decomplexation) or to build it (case of a complexation).

We finally observe that in the above reactions we have only represented the *active* sites in the left-hand sides, namely those sites which are tested and perhaps modified by the rule. The meaning of this is that all the other sites are kept intact by the reaction. This convenient notation is introducing some element of pattern matching or evaluation context in our operational semantics. While it seems not a big deal for the computer scientist, to our knowledge, no direct biological modeling language is using even this simple form of contextual operational semantics.

3. The κ-calculus

We now develop an algebraic notation as a workable syntax for our graphs-with-sites introduced in the previous section. One might wonder why such an alternative notation is needed or even useful.

One point in favor of this new notation is that it gives a precise description in the classical style of π-calculus [20] and this could be a beginning leading to nice reasoning principles. Having the name creator, "new", in the syntax allows for a clean syntactic treatment of edge creations in the right hand sides of reactions. One does not have to bother with freshness conditions when defining the operational semantics: name creation and structural congruence do it themselves.

A side observation perhaps only of interest for Concurrency theorists is that of the ordinary material of process algebras, we just use here parallel composition and name creation, a pretty minimal algebraic subset. Communication is done à la Join [12], by means of reactions, but with an additional edge structure on messages which is imposed by reactions. A minor advantage that comes with such a traditional notation is that there are tools specifically meant to manipulate π-like syntactic structures, as in a language recently proposed by Cardelli, Gardner and Ghelli [4], which would provide natural environments for the development of models in κ.

A second point is that having both our target intermediate language $m\kappa$, later to be compiled in π-calculus, and the source language displayed in the same syntactic styles eases the work of translating one in the other.

A last point is that once this new notation is in place, one may consider more advanced notions of rewriting, such as hypergraph rewriting or interaction net rewriting [19], and explore this further. For instance, by asking whether such advanced rewritings are translatable in $m\kappa$, or in other words which of them may be implemented by means of binary interactions. Though we do not do this here, this seems worth pursuing.

Of course, there is a point against this syntax, namely that it is not as intuitive as the visual one we started with and this is why we introduced the visual notation first.

3.1. The syntax of κ

The syntax of κ-calculus relies on:

- a countable set of *protein names* $\mathcal{P}$, ranged over by A, B, $C,\ldots$
- a countable set of *edge names* $\mathcal{E}$, ranged over by x, y, $z,\ldots$
- a *signature* map, written $\mathfrak{s}$, from $\mathcal{P}$ to natural numbers $\mathbb{N}$.

For each protein name A, $\mathfrak{s}(A)$ is the number of sites of A, and for any $1 \leqslant i \leqslant \mathfrak{s}(A)$, the pair (A,i) will accordingly be called a *site* of A.

Interfaces. An *interface* is a partial map from $\mathbb{N}$ to $\mathcal{E}+\{h,v\}$ usually ranged over by ρ, σ and similar symbols. The domain and range of an interface ρ will be respectively denoted by $\mathrm{dom}(\rho)$ and $\mathrm{ran}(\rho)$, and the set of names free in ρ, written $\mathrm{fn}(\rho)$, is obtained as $\mathrm{ran}(\rho)\cap\mathcal{E}$. We will only ever deal with interfaces with finite domain. The empty interface will be denoted $\varnothing$.

A site (A,i) of A is said to be *visible* in an interface ρ if $\rho(i)=v$, *hidden* if $\rho(i)=h$, *free* if it is visible or hidden and *bound* if $\rho(i)\in\mathcal{E}$. Any interface ρ uniquely

Table 1
The syntax of κ-calculus

S :=	**solution**
0	empty solution
$A(\rho)$	protein
S,S	group
$(x)(\mathsf{S})$	new

decomposes as a disjoint sum $\rho_1+\rho_2$ where $\mathrm{ran}(\rho_1)\subseteq\mathcal{E}$, $\mathrm{ran}(\rho_2)\subseteq\{h,v\}$, ρ_2 will be called the *free interface* of ρ. Interfaces are used to depict partial states of A's sites. The state depicted by ρ may be only partial since $\mathrm{dom}(\rho)$ may not contain the whole of $\mathfrak{s}(A)$.

We could have called ρ a state instead of an interface. This choice of terminology is meant to insist on the fact that the interface is going to determine the interaction capabilities of the protein it is an interface of.

Let us have an example. If A is such that $\mathfrak{s}(A)=3$ then $\rho(1)=v$, $\rho(2)=h$, $\rho(3)=x$ is a well-defined interface map for A, that declares site 1 to be visible, site 2 to be hidden and site 3 to be bound to some name x. We will write simply $\rho=1+\bar{2}+3^x$. Take note that in this way of writing things, the operation "+" represents a disjoint sum and indeed all terms in the sum have disjoint domains.

Proteins and solutions: The syntax given in Table 1 defines a *solution*, which can be either the empty solution, or a *protein* $A(\rho)$ with $A\in\mathcal{P}$ and ρ an interface with domain $\mathfrak{s}(A)$, or a *group* of solutions S, S', or a solution prefixed by a *new* name constructor $(x)(\mathsf{S})$ with $x\in\mathcal{E}$.

A convenient abbreviation will be to write $(x_1\cdots x_n)(\mathsf{S})$ or even sometimes $(\tilde{x})(\mathsf{S})$ instead of $(x_1)\cdots(x_n)(\mathsf{S})$.

The "new" operator is a binder: in $(x)(\mathsf{S})$, S is the *scope* of the binder (x). One inductively defines the set $\mathrm{fn}(\mathsf{S})$ of *free names* in a solution S:

$$\begin{aligned}\mathrm{fn}(0) &= \varnothing,\\ \mathrm{fn}(A(\rho)) &= \mathrm{fn}(\rho),\\ \mathrm{fn}(\mathsf{S},\mathsf{S}') &= \mathrm{fn}(\mathsf{S})\cup\mathrm{fn}(\mathsf{S}'),\\ \mathrm{fn}((x)(\mathsf{S})) &= \mathrm{fn}(\mathsf{S})\backslash\{x\}.\end{aligned}$$

An occurrence of x in S is *bound* if it occurs in a sub-solution which is in the scope of a binder (x); a solution S is *closed* if all occurrences of names in S are bound or equivalently if $\mathrm{fn}(\mathsf{S})=\varnothing$.

For instance, in

$$\mathsf{S}=C(1^x+2),(x)(A(1^x+2+\bar{3}),B(1+2^x))$$

both occurrences of x in A and B are bound, while the occurrence in C is outside the scope of (x) and hence is not bound in S. In particular, $\mathrm{fn}(\mathsf{S})=\{x\}$, and S is not closed.

3.2. Structural congruence

While our notation is certainly precise, it is also very rigid because it separates solutions that we do not want to distinguish for any semantic reason. Therefore we introduce an equivalence relation between solutions, called the *structural congruence*.

Definition 1. Structural congruence, written $\equiv$, is the least equivalence closed under syntactic constructions, containing α-equivalence (injective renaming of bound variables), taking "," to be associative (as the choice of symbol suggests) and commutative, with $\mathsf{0}$ as neutral element, and satisfying the scope laws:

$$\begin{array}{rcll} (x)(y)(\mathsf{S}) & \equiv & (y)(x)(\mathsf{S}), & \\ (x)(\mathsf{S}) & \equiv & \mathsf{S} & \text{when } x \notin \mathrm{fn}(\mathsf{S}), \\ (x)(\mathsf{S}),\ \mathsf{S}' & \equiv & (x)(\mathsf{S},\ \mathsf{S}') & \text{when } x \notin \mathrm{fn}(\mathsf{S}'). \end{array}$$

Coming back to the example above, the reader might want to check that:

$$\mathsf{S} \equiv (y)(C(1^x + 2), A(1^y + 2 + \bar{3}), B(1 + 2^y)) = \mathsf{T},$$

and one can observe that $\mathrm{fn}(\mathsf{S}) = \mathrm{fn}(\mathsf{T})$. This property holds in general, namely free names are invariant under structural equivalence. This because, intuitively, equivalent solutions describe the same underlying object.

3.3. Graph-likeness

So far we have a language that can describe more general objects than just graphs-with-sites. For instance one may write:

$$(x)(A(1^x)) \quad \text{or} \quad (x)(A(1^x), A(1^x), B(1^x)).$$

These might be interesting to study and, as said, have a natural interpretation as hypergraphs. Reactions defined on such terms would encode some sort of hypergraph-rewriting. But we are not primarily interested in them in this paper, since our concern is to home in on a simple notation that will be expressive enough for representing biological interaction, but no more.

Definition 2 (graph-likeness). A solution S is said to be graph-like if:
– free names occur at most twice in S;
– binders in S bind either zero or two occurrences.

If in addition free names occur exactly twice in S, we say that S is strongly graph-like.

Bound names are supposed to represent edges, and an edge has two endpoints, so the second condition speaks for itself. The first condition is just what one needs to cope with solutions with free names.

We take note that we must check at some point later that reactions, which have yet to be defined properly, preserve graph-likeness.

It is worth pointing out the relationship between our graph-with-sites of the preceding section and the algebraic notation developed here. The following translation uses, as intermediate constructs, graphs where some sites are labeled by names.

Definition 3. Let $[\![\cdot]\!]_g$ be the following function from graph-like solutions to graphs with sites:

1. $[\![A(\rho)]\!]_g$ is the graph with a single node labeled A, sites in $\{1,\ldots,\mathfrak{s}(A)\}$, bound sites k being labeled by $\rho(k)$, and free sites being in the state prescribed by ρ;
2. $[\![\mathsf{S},\mathsf{S}']\!]_g$ is the union graph of $[\![\mathsf{S}]\!]_g$ and $[\![\mathsf{S}']\!]_g$, with sites labeled with the same name being connected by an edge, and their common name erased;
3. $[\![(x)(\mathsf{S})]\!]_g$ is $[\![\mathsf{S}]\!]_g$.

It is easy to see that if two solutions S, S' are closed and graph-like, then $\mathsf{S}\equiv\mathsf{S}'$ if and only if $[\![\mathsf{S}]\!]_g = [\![\mathsf{S}']\!]_g$. So that the meaning of the structural equivalence on graph-like solutions is clear: it is equivalent to denoting the same graph-with-sites.

One can also turn a graph into a closed graph-like solution. Informally, each node becomes a protein in the solution and for each edge a fresh name is put at the edge ends on the appropriate sites, then the whole expression is closed by as many "new" operators as there are edges. Again it is easy to see that this second construction is inverse, up to $\equiv$, to the one above on closed graph-like solutions.

Thus, our term calculus is really a textual notation for graphs in the case of graph-like solutions. For instance, using structural congruence one can define connectedness and complexes:

- $A(\rho)$ is connected;
- if S is connected so is $(x)(\mathsf{S})$;
- if S and S' are connected and $\mathrm{fn}(\mathsf{S})\cap\mathrm{fn}(\mathsf{S}')\neq\varnothing$ then S,S' is connected;
- if S is connected and $\mathsf{S}\equiv\mathsf{T}$ then T is connected.

A *complex* is then a closed connected graph-like solution and it can be readily proved that S is a complex if and only if $[\![\mathsf{S}]\!]_g$ is. This can be seen in the examples of Fig. 2, which correspond to the following terms:

$$(x)(A(1^x + 2^x + 3 + \bar{4})),$$
$$(wxyz)(A(1^x + 2^y + 3), B(1^z + \bar{2} + 3^y), C(1 + \bar{2} + 3^z + 4^w), D(1^w + 2^x)),$$
$$(xy)(A(\bar{1} + 2 + 3^x + 4^y), B(1 + \bar{2} + 3^y + 4^x)).$$

3.4. Biological reactions

To keep track of interfaces, we now construct the *growth relation* on partial interfaces. This relation is parameterized by a set of names, written $\tilde{x}$ below, which represent (a superset of) edges grown out of a reaction. It is written $\leqslant$ and is defined inductively by the clauses given in Table 2.

Suppose one can derive $\tilde{x}\vdash\rho\leqslant\sigma$, then according to the (switch) clauses, σ may toggle free sites from visible to hidden, while according to the (create) clause, σ may only bind sites that were formerly *visible* in ρ. This makes formal the intuition that hidden sites are not accessible for binding and have to be made visible in one way or

Table 2
The growth relation

$$\textsf{create}\ \frac{x \in \widetilde{x}}{\widetilde{x} \vdash \iota \leqslant \iota^x}$$

$$\textsf{hv–switch}\ \frac{}{\widetilde{x} \vdash \bar{\iota} \leqslant \iota} \qquad \frac{}{\widetilde{x} \vdash \iota \leqslant \bar{\iota}}\ \textsf{vh–switch}$$

$$\textsf{reflex}\ \frac{\widetilde{x} \cap \mathrm{fn}(\rho) = \varnothing}{\widetilde{x} \vdash \rho \leqslant \rho} \qquad \frac{\widetilde{x} \vdash \rho \leqslant \sigma \qquad \widetilde{x} \vdash \rho' \leqslant \sigma'}{\widetilde{x} \vdash \rho + \rho' \leqslant \sigma + \sigma'}\ \textsf{sum}$$

Table 3
Extended growth relation

$$\textsf{nil}\ \frac{}{\widetilde{x} \vdash 0 \leqslant 0} \qquad \frac{\widetilde{x} \vdash \mathsf{S} \leqslant \mathsf{T} \qquad \widetilde{x} \vdash \rho \leqslant \sigma \qquad \mathrm{dom}(\sigma) \subseteq \mathfrak{s}(A)}{\widetilde{x} \vdash \mathsf{S}, A(\rho) \leqslant \mathsf{T}, A(\sigma)}\ \textsf{group}$$

$$\frac{\widetilde{x} \vdash \mathsf{S} \leqslant \mathsf{T} \qquad \mathrm{fn}(\sigma) \subseteq \widetilde{x} \qquad \mathrm{dom}(\sigma) = \mathfrak{s}(A)}{\widetilde{x} \vdash \mathsf{S} \leqslant \mathsf{T}, A(\sigma)}\ \textsf{synth}$$

another to become available. It is therefore important that $\leqslant$ is not a transitive relation, despite the notation! Else, from $\bar{\iota} \leqslant \iota$ and $\iota \leqslant \iota^x$ one would deduce the unintended $\bar{\iota} \leqslant \iota^x$ and get access to hidden sites.

We also remark that:
- $\mathrm{dom}(\sigma) = \mathrm{dom}(\rho)$, that is both σ and ρ must have the same domain,
- sites bound by ρ cannot be freed by σ,
- and created edges have to belong to $\tilde{x}$ and be separated from names used by ρ as specified by the (reflex) clause. Typically, one has $y \vdash 1^x + 2 \leqslant 1^x + 2^y$ but $\tilde{x} \nvdash 1^x + 2 \leqslant 1^x + 2^x$.

A partial interface with range in $\{h, v\}$, will be related to any other partial interface with the same domain.

When writing down biological reactions, it is of great convenience to address only the part of proteins that are changed or checked during the reaction, rather than specifying the whole interface. So we define $A(\sigma)$ to be a *pre-protein* if σ is a partial interface of A, namely $\mathrm{dom}(\sigma) \subseteq \mathfrak{s}(A)$. Similarly we define pre-solutions as combinations of pre-proteins, obtained as in Table 1. That said, we can extend the growth relation to groups of pre-proteins as shown in Table 3. Take note that this definition only applies to pre-solutions without any "new". Growing means creating edges and possibly creating proteins as well, as in the (synth) clause. Observe that this clause requires the newly created protein A to have a *complete* interface, that is an interface σ such that $\mathrm{dom}(\sigma) = \mathfrak{s}(A)$ and also asks that all edges in A are new, and hence have their names in $\tilde{x}$.

Lemma 1. *Let* L, R *be two pre-solutions such that* $\tilde{x} \vdash \mathsf{L} \leqslant \mathsf{R}$, *then* $\mathrm{fn}(\mathsf{L}) = \mathrm{fn}(\mathsf{R}) \backslash \tilde{x}$ *and* $\mathrm{fn}(\mathsf{R}) \subseteq \mathrm{fn}(\mathsf{L}) + \tilde{x}$.

Proof. A first easy induction on the growth relation shows the analog statement for interfaces, that is if $\tilde{x} \vdash \rho \leqslant \sigma$, then $\mathrm{fn}(\rho) = \mathrm{fn}(\sigma) \backslash \tilde{x}$, and further $\mathrm{fn}(\sigma) \subseteq \mathrm{fn}(\rho) + \tilde{x}$ and then a second induction shows that this is preserved by the pre-solution extension. □

The purpose of the (synth) clause, as we will see below, is to express synthesis mechanisms and by using the dual relation $\geqslant$, to express degradation as well.

Definition 4. Let L, R be two pre-solutions,

- $\mathsf{L} \to (\tilde{x})\mathsf{R}$ is said to be a *monotonic reaction* if:
  - $\tilde{x} \vdash \mathsf{L} \leqslant \mathsf{R}$,
  - both L and $(\tilde{x})\mathsf{R}$ are graph-like,
  - and R is connected.
- $(\tilde{x})\mathsf{L} \to \mathsf{R}$ is said to be an *anti-monotonic reaction* if:
  - its dual $\mathsf{R} \to (\tilde{x})\mathsf{L}$ is monotonic.

A reaction which is either monotonic or antimonotonic is called a *biological* reaction and L and R are referred to, respectively, as its *reactants* and *products*.

A direct consequence of the lemma above is that free names are preserved, i.e., $\mathrm{fn}(\mathsf{L}) = \mathrm{fn}((\tilde{x})\mathsf{R})$, in biological reactions. So it makes sense to refer to these common free names as the free names of the reaction $\mathfrak{r}$, denoted by $\mathrm{fn}(\mathfrak{r})$. Actually, the growth condition alone, $\tilde{x} \vdash \mathsf{L} \leqslant \mathsf{R}$, makes sure that these common free names are used to connect the *same* sites in the reactants and the products. [1] As a consequence, edges, which are created in a monotonic reaction, or deleted in an antimonotonic one, have to be syntactically bound.

Just to recap this *basic principle*:

– bound names correspond to created (resp. deleted) edges,
– free names correspond to edges left intact.

Reactions are syntactically reversible so that one could define only monotonic reactions and embed reversibility in the definition of the transition system below. But it seems more intuitive to have both kinds of reactions directly in the syntax.

The choice of a monotonic format for reactions embodies our postulate about protein interaction. This is in accordance with the detailed descriptions that biologists give of their systems and it also meshes with what is taken to be an atomic step in the kinetic analysis of biochemical reactions. Of course, one could argue that even these reactions are still not atomic enough and that, at a lower level, biology is blind and reactions have to be decomposed as binary interactions. And indeed this is the problem which

[1] The notion of *same* may depend on the derivation of $\mathsf{L} \leqslant \mathsf{R}$ in the presence of synthesis. If there are enough symmetries in L and R, there could be many derivations. Here is an example:

$$x, y \vdash A(1+2) \leqslant A(1^x + 2), A(1^y + \bar{2}), B(1^x), C(1^y)$$

which might be derived in two ways, depending on which of the As on the right is synthesized. The corresponding monotonic reaction when applied to $A(1+2)$ will give different results. Thus, to be completely accurate and get rid of this ambiguity, as would be in order for an implementation for instance, one would incorporate the derivation of $\mathsf{L} \leqslant \mathsf{R}$ in the definition of the reaction.

we address in Section 6. By the way, we could have constrained the reactions further by asking that the reactants and products are *strongly* graph-like. Actually none of the examples developed in the next section, devoted to biological systems, is seriously using the additional expressivity which our choice allows. Yet, when we explore the matter of self-assembly, this will turn out to be the right choice.

Relaxing the format. We could also have considered less constrained reactions such as the following:

$$(x)(A(1^x+2),B(1^x+2),C(1)) \to (z)(A(1+2),B(1+2^z),C(1^z))$$

which is mixing monotonic features, the edge z between B and C is created, and antimonotonic ones, the edge x between A and B is deleted. At a higher level of granularity, such reactions could be taken as basic as well, as they can be decomposed as a monotonic reaction followed by an antimonotonic one:

$$A(1^x+2),B(1^x+2),C(1) \to (z)(A(1^x+2),B(1^x+2^z),C(1^z))$$
$$(x)(A(1^x+2),B(1^x+2^z),C(1^z)) \to A(1+2),B(1+2^z),C(1^z)$$

Because the intermediate product which this decomposition is making explicit is connected, it seems reasonable to consider the sequence as a synchronous composition. The case of the following "edge-flipping" reaction, as we know from Section 2, is different:

$$(y)(A(1^x+2^y),B(1^x+2),C(1^y)) \to (z)(A(1^x+2),B(1^x+2^z),C(1^z)).$$

Since there is not a free site for C to bind with B, one cannot perform a similar decomposition. This reaction can only be decomposed as an antimonotonic reaction and a monotonic one:

$$\begin{aligned}(y)(A(1^x+2^y),B(1^x+2),C(1^y)) &\to A(1^x+2),B(1^x+2),C(1)\\ A(1^x+2),B(1^x+2),C(1) &\to (z)(A(1^x+2),B(1^x+2^z),C(1^z)).\end{aligned}$$

Now, there is no intermediate connected product, and thus no guarantee that these two reactions will actually be applied in a sequence. Such a synchronization needs a specific control mechanism.

3.5. Biological transition systems

A *renaming* r is a finite partial injection on $\mathcal{E}+\{h,v\}$, which is the identity on $\{h,v\}$ and maps $\mathcal{E}$ into $\mathcal{E}$.

Definition 5 (matching). Given a monotonic reaction $\mathsf{L}\to(\tilde{x})\mathsf{R}$, with:
– $\mathsf{L}=A_1(\rho_1),\ldots,A_n(\rho_n)$
– and $\mathsf{R}=A_1(\sigma_1),\ldots,A_m(\sigma_m)$,
one says that a pair of solutions S, T *matches* $\mathsf{L}\to(\tilde{x})\mathsf{R}$, written $\mathsf{S},\mathsf{T}\models\mathsf{L}\to(\tilde{x})\mathsf{R}$, if there exists a renaming r and partial interfaces $\xi_1,\ldots,\xi_m$ such that:
1. for all i, $r(\tilde{x})\cap \mathrm{fn}(\xi_i)=\varnothing$,

2. $\mathsf{S} = A_1(r \circ \rho_1 + \xi_1), \ldots, A_n(r \circ \rho_n + \xi_n)$ and $\mathsf{T} = (r(\tilde{x}))(A_1(r \circ \sigma_1 + \xi_1), \ldots, A_m(r \circ \sigma_m + \xi_m))$.

Matching is defined by symmetry for antimonotonic rules, that is $\mathsf{S}, \mathsf{T} \models (\tilde{x})\mathsf{L} \to \mathsf{R}$ if and only if $\mathsf{T}, \mathsf{S} \models \mathsf{R} \to (\tilde{x})\mathsf{L}$.

The first condition makes sure that none of the created names is used by the interfaces extensions ξ_is.[2] The second condition is merely saying that under such renaming and extensions, L and R instantiate to S and T.

It is worth mentioning that, since S, T are solutions, all the interfaces $r \circ \rho_i + \xi_i$, $r \circ \sigma_i + \xi_i$ have to be complete ones, and therefore:

$$\mathrm{dom}(r) \supseteq \bigcup_i \mathrm{fn}(\sigma_i) \supseteq \bigcup_i \mathrm{fn}(\rho_i)$$

with the second inclusion given by monotonicity.

A more abstract and equivalent view of matching is possible. Derivations, as defined in Tables 2 and 3, can themselves be renamed by injective renaming and extended by picking larger interfaces in (reflex) clauses. This defines an "instantiation" preorder between derivations and S, $(\tilde{y})\mathsf{T}$ can then be defined to match $(\tilde{x})\mathsf{L} \to \mathsf{R}$ if both $\tilde{y} \vdash \mathsf{S} \leqslant \mathsf{T}$ and $\tilde{x} \vdash \mathsf{L} \leqslant \mathsf{R}$ can be derived in such a way that the derivation of the former is below that of the latter, according to this preorder. The first condition in the definition above is taken care of by the (reflex) clause, while the second one is automatically satisfied, because of the preorder.

Lemma 2. *Let* $\mathsf{L} \to (\tilde{x})\mathsf{R}$ *be a monotonic reaction and* S, T *a pair of matching solutions, then* (1) *occurrences of free names are in bijection between* S *and* T; (2) S *is graph-like if and only if* T *is.*

Proof. Suppose first S is graph-like and consider a name x occurring in T. If $x \notin \mathrm{fn}(\mathsf{T})$, then $x \in r(\tilde{x})$, and by the first condition: $x \notin \bigcup_i \mathrm{fn}(\xi_i)$, so that its only occurrences come from R along the partial injection r, and since $(\tilde{x})\mathsf{R}$ is graph-like, this means $r^{-1}(x)$ has exactly two occurrences in R, and therefore also two occurrences in T, as it should. If else $x \in \mathrm{fn}(\mathsf{T})$, then none of its occurrences is created, i.e., introduced by the (create) clause or the (synth) clause (because this clause asks that all names introduced are in $\tilde{x}$), in the derivation of $\tilde{x} \vdash \mathsf{L} \leqslant \mathsf{R}$, therefore all must be inherited from the (reflex) clause or provided by an interface extension ξ_i, and in both cases the same occurrences exist in S and not more, since names cannot be deleted in a monotonic reaction.

Suppose conversely that T is graph-like and x occurs in S. Then $x \in \mathrm{fn}(\mathsf{S})$, since no name is bound in S, and occurrences of x are either provided by a (reflex) clause or an interface extension ξ_i, in both cases the same occurrences exist in T and not more, since other occurrences would have to be created and any name created in a monotonic reaction is bound. □

[2] This would result otherwise in a non-graph-like T since created names are bound and $(\tilde{x})\mathsf{R}$ being graph-like they must appear exactly twice, so if ξ_i uses one of the $r(\tilde{x})$ there will be at least three occurrences of a same name in T. So one does not really have to ask this when dealing only with graph-like solutions, but it seems clearer to do so and prove just below that our format given in Definition 4 respects graph-likeness.

We observe that the argument does not use the connectedness assumption on R, which is there for completely different reasons explained at length in the preceding section.

Examples of matching: Here is a match on a monotonic reaction with $r(x)=u$, $\xi_1=3^z$, $\xi_2=\bar{2}$:

$$\frac{A(1+2),B(1)\to(x)(A(1^x+2),B(1^x))}{A(1+2+3^z),B(1+\bar{2})\to(u)(A(1^u+2+3^z),B(1^u+\bar{2}))}.$$

A somewhat subtler example, where r must deal with both the free and the bound variable, $r(u)=x$, $r(x)=y$, $\xi_1=\bar{3}$, $\xi_2=\varnothing$:

$$\frac{A(1+2^u),B(1+2^u)\to(x)(A(1^x+2^u),B(1^x+2^u))}{A(1+2^x+\bar{3}),B(1+2^x)\to(y)(A(1^y+2^x+\bar{3}),B(1^y+2^x))}.$$

Notice also that while ξ_i and r may not overlap on bound names, they may do so on free names as in the following match:

$$\frac{A(1^x+2),B(1)\to(z)(A(1^x+2^z),B(1^z))}{A(1^x+2+3^x),B(1)\to(z)(A(1^u+2^z+3^x),B(1^z))},$$

where $r(x)=x=\xi_1(3)$.

Definition 6. Let $\mathfrak{R}$ be a set of biological reactions, the associated $\mathfrak{R}$-*system* is the pair $(\mathfrak{S},\to)$, where $\mathfrak{S}$ is the set of solutions and $\to$, called the *transition* relation, is the least binary relation over $\mathfrak{S}$ such that:

$$\text{mon}\ \frac{\mathsf{S},\mathsf{T}\models\mathsf{L}\to(\tilde{\mathsf{x}})\mathsf{R}\ \in\mathfrak{R}}{\mathsf{S}\to\mathsf{T}} \qquad \frac{\mathsf{S},\mathsf{T}\models(\tilde{\mathsf{x}})\mathsf{L}\to\mathsf{R}\ \in\mathfrak{R}}{\mathsf{S}\to\mathsf{T}}\ \text{antimon}$$

$$\text{new}\ \frac{\mathsf{S}\to\mathsf{T}}{(x)(\mathsf{S})\to(x)(\mathsf{T})} \qquad \frac{\mathsf{S}\to\mathsf{T}}{\mathsf{S},\mathsf{S}'\to\mathsf{T},\mathsf{S}'}\ \text{group}$$

$$\text{struct}\ \frac{\mathsf{S}\equiv\mathsf{S}'\quad\mathsf{S}'\to\mathsf{T}'\quad\mathsf{T}'\equiv\mathsf{T}}{\mathsf{S}\to\mathsf{T}}$$

Contextual rules allow to focus on the reacting parts of the system. With this definition, we may give an example of what goes wrong when one violates the side-condition in the (create) clause:

$$\text{group}\ \frac{\text{mon}\ \dfrac{}{A(1^y+2)\to A(1^y+2^y)}}{A(1^y+2),B(1^y)\to A(1^y+2^y),B(1^y)}$$

resulting in a non-graph-like right hand side. If everything was done properly in the definition of reactions one should be able to extend Lemma 2 and show that the example above never happens with proper reactions.

Proposition 3. *Suppose* $\mathsf{S}\to\mathsf{T}$ *then* (1) *occurrences of free names are in bijection between* S *and* T; (2) S *is graph-like if and only if* T *is.*

Proof. The basic case corresponds to Lemma 2 so it remains to prove that the three contextual rules preserve our property. The (new) rule clearly does, since $(x)\mathsf{S}$ is

graph-like iff S is and x occurs twice or not at all in S. The (struct) rule also does since both the properties of being graph-like and of being a free name occurrence are invariants of structural equivalence, as noticed earlier. Finally there is the (group) rule, which obviously preserves both conditions. □

So our reactions and the accompanying transition systems preserve graph-likeness, and it makes sense to restrict to graph-like solutions which is what we are going to do from now on.

4. Formal biological systems

With a well-defined language of idealized protein interactions, the next thing one needs is some examples to measure how well the language performs in the description of typical biological systems.

The first example is all about protein–protein interaction and our language passes that expressivity test with no problem. The second example is richer and the simple model we obtained is meant as an assessment of what our language in the present stage can do and how well it can do it. Further modeling practice will refine the picture and give a better sense of which extensions are the most needed.

4.1. Signals

The first steps of the signal cascade triggered by the growth factor EGF are detailed enough that we can give a minute description of what is going on: a dimeric form EGF_2 of the growth factor EGF binds two receptors EGFR (also known as RTK); the receptors cross-phosphorylate each other through their tyrosine kinase sites; once this is done each can activate a second binding site and then bind an adapter protein SHC and activate it. The signal goes then further down and passes through many other proteins, but we stop our description here.

To keep things readable we will rename our protagonists as S the signal, R the receptor and A the adapter and after the biological description choose them of respective arities 2, 3, and 2. The particular site $(\mathsf{R},2)$ stands for the receptor tyrosine kinase site. Here is the formal rendering:

Signal–Receptor Interaction

$$\tau_1 : \mathsf{S}(1), \mathsf{S}(1) \to (x)(\mathsf{S}(1^x), \mathsf{S}(1^x))$$
$$\tau_2 : \mathsf{S}(2), \mathsf{R}(1) \to (x)(\mathsf{S}(2^x), \mathsf{R}(1^x))$$

RTK Cascade

$$\tau_3 : \mathsf{S}(1^x + 2^y), \mathsf{S}(1^x + 2^z), \mathsf{R}(1^y), \mathsf{R}(1^z + \bar{2}) \to \mathsf{S}(1^x + 2^y), \mathsf{S}(1^x + 2^z), \mathsf{R}(1^y), \mathsf{R}(1^z + 2)$$
$$\tau_4 : \mathsf{R}(2 + \bar{3}) \to \mathsf{R}(2 + 3)$$
$$\tau_5 : \mathsf{R}(3), \mathsf{A}(1) \to (x)(\mathsf{R}(3^x), \mathsf{A}(1^x))$$
$$\tau_6 : \mathsf{R}(2 + 3^x), \mathsf{A}(1^x + \bar{2}) \to \mathsf{R}(2 + 3^x), \mathsf{A}(1^x + 2)$$

Table 4
A run of the RTK cascade

$\mathsf{S}(1+2), \mathsf{S}(1+2), \mathsf{R}(1+\bar{2}+\bar{3}), \mathsf{R}(1+\bar{2}+\bar{3}), \mathsf{A}(1+\bar{2})$	
$\to (x)(\mathsf{S}(1^x+2), \mathsf{S}(1^x+2)), \mathsf{R}(1+\bar{2}+\bar{3}), \mathsf{R}(1+\bar{2}+\bar{3}), \mathsf{A}(1+\bar{2})$	(τ_1)
$\to (xy)(\mathsf{S}(1^x+2), \mathsf{S}(1^x+2^y), \mathsf{R}(1^y+\bar{2}+\bar{3})), \mathsf{R}(1+\bar{2}+\bar{3}), \mathsf{A}(1+\bar{2})$	(τ_2)
$\to (xyz)(\mathsf{S}(1^x+2^z), \mathsf{S}(1^x+2^y), \mathsf{R}(1^y+\bar{2}+\bar{3}), \mathsf{R}(1^z+\bar{2}+\bar{3})), \mathsf{A}(1+\bar{2})$	(τ_2)
$\to (xyz)(\mathsf{S}(1^x+2^z), \mathsf{S}(1^x+2^y), \mathsf{R}(1^y+2+\bar{3}), \mathsf{R}(1^z+\bar{2}+\bar{3})), \mathsf{A}(1+\bar{2})$	(τ_3)
$\to (xyz)(\mathsf{S}(1^x+2^z), \mathsf{S}(1^x+2^y), \mathsf{R}(1^y+2+3), \mathsf{R}(1^z+\bar{2}+\bar{3})), \mathsf{A}(1+\bar{2})$	(τ_4)
$\to (xyzu)(\mathsf{S}(1^x+2^z), \mathsf{S}(1^x+2^y), \mathsf{R}(1^y+2+3^u), \mathsf{R}(1^z+\bar{2}+\bar{3}), \mathsf{A}(1^u+\bar{2}))$	(τ_5)
$\to (xyzu)(\mathsf{S}(1^x+2^z), \mathsf{S}(1^x+2^y), \mathsf{R}(1^y+2+3^u), \mathsf{R}(1^z+\bar{2}+\bar{3}), \mathsf{A}(1^u+2))$	(τ_6)

The key constraint is that the dormant capacity of the $(\mathsf{R}, 2)$ site can only be woken up by reaction 3, and only then is the 4–6 cascade possible. With the reactions one can run a minimal interesting system, as shown in Table 4, starting in a quiescent state where 2 is hidden in R, else the receptor would be active right away, and 2 is hidden as well in A, else A would already be active.

Nature could perhaps have chosen a simpler design by letting the signal be itself an activator (a kinase) and not resorting to the receptor for activation. Indeed, in the solution above the receptor has to be in "suspended state" until its activation capability is triggered by the signal. Whether there is an other constraint on the design that makes this solution reasonable or it is just a matter of chance, we do not know. Be that as it may, one sees that the calculus expresses the causality involved in the transduction in a precise yet natural way.

4.2. The lactose operon

Let us turn now to a more comprehensive example. Escherichia Coli, one of the most studied organisms, has glucose (Glu) as the input of an important metabolic pathway, *glycolysis*, leading to the production of pyruvic acid and eventually of ATP which is the major energy currency in the cell. Sometimes there is not enough glucose and E. Coli has to feed on alternative food. If lactose (Lac) is around, E. Coli can trigger the synthesis of:

- galactosidase (GAL) which can turn lactose into glucose,
- and of a permease (PER) that helps the bigger lactose enter the cell.

Then lactose flows in and the cell is back into business. Yet there is need to control when this happens and when one may switch back to the ordinary behavior and feed again directly on glucose. The description of the molecular level implementation of the control was one of the major discoveries of early molecular biology [21]. For us this will be the occasion to review some typical molecular events, test the expressiveness of our language, and discuss some possible extensions.

Molecular control. But first we have to describe the molecular control in biological terms. Though as said, this particular system has been studied a lot, some questions are still open and here we will be happy with a somewhat simplified description. The reader curious to learn more may refer to Kimball's Biology Pages [16].

An *operon* is a sequence of genes which are transcribed collectively, together with a repressor protein that can block the transcription of the genes. The *Lac-operon* contains the genes coding respectively for GAL and PER and its repressor protein was aptly named REP. This is the device the cell wants to turn *on* to handle lactose and *off* if there is enough glucose. Upstream of this operon, there are three small regions on the DNA:

– a site where some complex CAP · cAMP can bind and have a positive influence on the operon transcription,
– the promoter site where RNAp the transcription machinery binds, and then opens the DNA helix and proceeds down one strand beginning the transcription process,
– and overlapping with the promoter, the so-called operator site, where REP can bind and therefore prevent the recruitment of RNAp, blocking the transcription of the whole operon and therefore the synthesis of the associated proteins.

In the absence of Lac, REP binds to DNA and switches off the operon. Likewise, in abundant presence of Glu, the production of cAMP is inhibited, therefore the complex CAP · cAMP does not bind, so again our operon is off. To turn it on, there must be a low level of Glu, so that CAP · cAMP binds. But this is not enough, one also needs a certain amount of Lac and enough of GAL so that some aLac is produced. This isomeric form of lactose binds to REP, changes the shape of the repressor and pries it out from the DNA, therefore activating the operon. Upon a sudden change from glucose to lactose in the environment, E. Coli will produce GAL until it reaches 2% of its mass, which is enormous considering that water accounts already for 70% of the total mass. Nothing is more important than food, it seems.

Formalization. Since we have mentioned all the different molecules involved, we may now turn to the reactions. These are presented in the direction in which they make the best sense with respect to the overall intended behavior, but they are all in fact reversible. To ease reading, sites are given explicit names and to keep things short we do not write the obvious synthesis reaction for REP and CAP, nor the degradation reactions of all the participating products.

Operon Synthesis

$$\begin{aligned}
\mathfrak{r}_{0a} \;:\; & \mathsf{UP}(\overline{\text{rep-s}} + \text{rnap-s}^{x} + \text{cap-s}^{y}), \mathsf{RNAp}(\text{up-s}^{x} + \text{syn1} + \text{syn2}) \to \\
& (zu)(\mathsf{UP}(\overline{\text{rep-s}} + \text{rnap-s}^{x} + \text{cap-s}^{y}), \mathsf{RNAp}(\text{up-s}^{x} + \text{syn1}^{z} + \text{syn2}^{u}), \\
& \mathsf{GAL}(\text{lac-s} + \text{s}^{z}), \mathsf{PER}(\text{lac-s} + \text{s}^{u})) \\
\mathfrak{r}_{0b} \;:\; & (yz)(\mathsf{RNAp}(\text{syn1}^{y} + \text{syn2}^{z}), \mathsf{GAL}(\text{lac-s} + \text{s}^{y}), \mathsf{PER}(\text{lac-s} + \text{s}^{z})) \to \\
& \mathsf{RNAp}(\text{syn1} + \text{syn2}), \mathsf{GAL}(\text{lac-s} + \text{s}), \mathsf{PER}(\text{lac-s} + \text{s})
\end{aligned}$$

The basic switching mechanism is expressed in 0*a*: synthesis begins only if the repressor is absent and the auxiliary CAP is present. To shorten the reaction, we actually only test that something is bound to the cap-s site (and therefore reaction 0*a* is graph-like but not strongly so, because y occurs only once on each side). If there were other products competing with CAP, then one would have to be specific about who is binding at cap-s. Both reactions 0a–0b could be composed in a single not monotonic reaction of the "good" kind (see the discussion about relaxing the format in the

preceding section).

Operon Control

$$\begin{aligned}
\mathfrak{r}_1 &: \mathsf{UP}(\text{rnap-s} + \text{rep-s}), \mathsf{REP}(\text{up-s}) \to (x)(\mathsf{UP}(\overline{\text{rnap-s}} + \text{rep-s}^x), \mathsf{REP}(\text{up-s}^x))\\
\mathfrak{r}_2 &: \mathsf{UP}(\text{rnap-s} + \text{rep-s}), \mathsf{RNAp}(\text{up-s}) \to (x)(\mathsf{UP}(\overline{\text{rep-s}} + \text{rnap-s}^x), \mathsf{RNAp}(\text{up-s}^x))\\
\mathfrak{r}_3 &: \mathsf{UP}(\text{cap-s}), \mathsf{CAP}(\text{up-s} + \text{camp-s}^x), \mathsf{cAMP}(\text{cap-s}^x) \to\\
&\quad (z)(\mathsf{UP}(\text{cap-s}^z), \mathsf{CAP}(\text{up-s}^z + \text{camp-s}^x), \mathsf{cAMP}(\text{cap-s}^x)).
\end{aligned}$$

Whether CAP binds or not is independent of the occupancy state of the other sites of UP. On the other hand, REP and RNAp are hiding each other's sites in 1–2, and therefore are mutually exclusive.

Regulations

$$\begin{aligned}
\mathfrak{r}_4 &: \mathsf{CAP}(\text{camp-s}), \mathsf{cAMP}(\text{cap-s}) \to (x)(\mathsf{CAP}(\text{camp-s}^x), \mathsf{cAMP}(\text{cap-s}^x))\\
\mathfrak{r}_{5a} &: \mathsf{REP}(\text{up-s} + \text{alac-s}), \mathsf{aLac}(\text{rep-s}) \to (x)(\mathsf{REP}(\overline{\text{up-s}} + \text{alac-s}^x), \mathsf{aLac}(\text{rep-s}^x))\\
\mathfrak{r}_{5b} &: \mathsf{UP}(\text{rep-s}^x), \mathsf{REP}(\text{up-s}^x + \text{alac-s}), \mathsf{aLac}(\text{rep-s}) \to\\
&\quad (y)(\mathsf{UP}(\text{rep-s}^x), \mathsf{REP}(\text{up-s}^x + \text{alac-s}^y), \mathsf{aLac}(\text{rep-s}^y))\\
\mathfrak{r}_{5c} &: (x)(\mathsf{UP}(\text{rep-s}^x), \mathsf{REP}(\text{up-s}^x + \text{alac-s}^y), \mathsf{aLac}(\text{rep-s}^y)) \to\\
&\quad \mathsf{UP}(\text{rep-s}), \mathsf{REP}(\overline{\text{up-s}} + \text{alac-s}^y), \mathsf{aLac}(\text{rep-s}^y).
\end{aligned}$$

In most descriptions aLac is said to be able to complex with REP, even after REP has landed on DNA, and then pry it out from the DNA. This is expressed by means of the complexation and subsequent decomplexation 5b–5c. Be it in this way or directly by reaction 5a, REP is made inert by the hiding of its binding capability up-s.

PER and GAL activity

$$\begin{aligned}
\mathfrak{r}_6 &: \mathsf{PER}(\text{lac-s}), \mathsf{Lac}(\text{per-s} + \overline{\text{in}}) \to (x)(\mathsf{PER}(\text{lac-s}^x), \mathsf{Lac}(\text{per-s}^x + \text{in}))\\
\mathfrak{r}_7 &: (x)(\mathsf{PER}(\text{lac-s}^x), \mathsf{Lac}(\text{per-s}^x)) \to \mathsf{PER}(\text{lac-s}), \mathsf{Lac}(\text{per-s})\\
\mathfrak{r}_8 &: \mathsf{GAL}(\text{lac-s}), \mathsf{Lac}(\text{in} + \text{gal-s}) \to (x)(\mathsf{GAL}(\text{lac-s}^x), \mathsf{Lac}(\text{in} + \text{gal-s}^x))\\
\mathfrak{r}_{9a} &: (x)(\mathsf{GAL}(\text{lac-s}^x + \overline{\text{loaded}}), \mathsf{Lac}(\text{gal-s}^x)) \to \mathsf{GAL}(\text{lac-s} + \text{loaded})\\
\mathfrak{r}_{9b} &: \mathsf{GAL}(\text{loaded}) \to \mathsf{GAL}(\overline{\text{loaded}}), \mathsf{Glu}(\text{s}), \mathsf{Gal}(\text{rep-s})\\
\mathfrak{r}_{9c} &: \mathsf{GAL}(\text{loaded}) \to \mathsf{GAL}(\overline{\text{loaded}}), \mathsf{aLac}(\text{rep-s}).
\end{aligned}$$

Reaction 6 has the effect that Lac is now inside the cell. We encode this by using the site in as a state. With a first-class notion of membrane, written m[...] below, we could replace 6 and 8 with:

PER: a membrane-variant

$$\begin{aligned}
\mathfrak{r}'_6 &: \mathsf{m}[\mathsf{PER}(\text{lac-s})], \mathsf{Lac}(\text{per-s}) \to \mathsf{m}[(x)(\mathsf{PER}(\text{lac-s}^x), \mathsf{Lac}(\text{per-s}^x))]\\
\mathfrak{r}'_8 &: \mathsf{GAL}(\text{lac-s}), \mathsf{Lac}(\text{gal-s}) \to (x)(\mathsf{GAL}(\text{lac-s}^x), \mathsf{Lac}(\text{gal-s}^x))
\end{aligned}$$

and have a direct account of what PER is doing. Extending κ-calculus with membranes seems a good idea, and we are looking forward to adapt existing membrane calculi [1,3] to do this.

The three reactions 9a–9c are decomposing the action of the beta-galactosidase enzyme. Our monotonicity principle forces a bit of gymnastics here, as we have to introduce the intermediate and somewhat imaginary state GAL(loaded) where the enzyme is loaded with its metabolite but has not yet decided what to do with it, either two

smaller sugars Glu and Gal[3] as in 9b or an isomeric form as in 9c. Perhaps a more direct solution would be to write the alternative reactions:

$$\mathfrak{r}'_{9a} : (x)(\mathsf{GAL}(\text{lac-s}^x), \mathsf{Lac}(\overline{\text{per-s}} + \text{gal-s}^x)) \to \mathsf{GAL}(\text{lac-s}), \mathsf{Glu}(\text{s}), \mathsf{Gal}(\text{s})$$
$$\mathfrak{r}'_{9b} : (x)(\mathsf{GAL}(\text{lac-s}^x), \mathsf{Lac}(\overline{\text{per-s}} + \text{gal-s}^x)) \to \mathsf{GAL}(\text{lac-s}), \mathsf{aLac}(\text{rep-s})$$

and consider that sugars and metabolites are not taken into account in monotonicity constraints. This is a very mild adaptation of our language since it amounts to adding a new inference rule for the growth relation which applies only when M is a metabolite:

$$\text{metab}\ \frac{\tilde{x} \vdash \mathsf{S} \leqslant \mathsf{T} \qquad \text{fn}(\rho) = \varnothing \qquad \text{dom}(\rho) = \mathfrak{s}(M)}{\tilde{x} \vdash \mathsf{S}, M(\rho) \leqslant \mathsf{T}}.$$

This makes good sense in that these small molecules have a very different set of bio-chemical interactions than the larger proteins.

The reader might wonder what has become of the mechanism by which Glu reduces the cAMP-level. This mechanism is not known and it is even a matter of discussion whether the whole "cAMP model" is correct [15]. This points out a shortcoming of our language if one wants to use it to build actual models: unless one is given the explicit molecular mechanism, one is not able to incorporate the knowledge in the model.

5. The *mκ*-calculus

We turn to the issue of implementing the κ-calculus, and discuss a distributed implementation based on agents exchanging channel names during rendez-vous communications. We first present *mκ-calculus*, a finer-grained language which describes a less idealized formal biology.

5.1. Agents and solutions

Since one wants to decentralize the κ-systems it is natural to put more intelligence in the agents. Indeed the syntax of *mκ*-calculus is the same as for κ-calculus discussed in Section 3, except that our basic components, which we now call *agents*, have more information at their disposal. Each site is given an additional state to the effect that the agent can log what's up on this connection. To emphasize the process nature of *mκ*-calculus, their restricted form of reactions will be called *interactions*.

As names for agents we keep the same set of names, namely $\mathcal{P}$, and the same definition of signature as we had in κ. In addition to the edge names $\mathcal{E}$, we need a countable set $\mathcal{G}$ of *group names* ranged over by r, $r', \ldots$ to be used later as the means to build transient cooperative structures in our low-level systems. Edge and group names are supposed to be disjoint.

An *extended interface*, is a finite map from $\mathbb{N}$ to $(\mathcal{E} + \mathcal{G} + \{h, v\}) \times \mathbb{N}$, ranged over by θ and similar symbols. The integer part of $\theta(i)$ is referred to as a *log*.

[3] One should not confuse the sugar galactose Gal and the protein GAL. As a rule, in this example, we use uppercase for proteins and capitalized lowercase for metabolites. This is following the biological tradition.

An *agent* is a pair written $A(\theta)$ with $A \in \mathcal{P}$ and θ an extended interface defined on $\mathfrak{s}(A)$.

Extended interfaces are written with the same lightweight notation used previously in κ, e.g., if $\mathfrak{s}(A)=3$, and $\theta(1)=x,1$, $\theta(2)=r,5$ and $\theta(3)=h,0$, then one may simply write $A(1^{x,1}+2^{r,5}+\tilde{3}^{0})$. We will also indulge sometimes in not writing a log when it is 0, so that for instance $C(1^{x,4}+2^{r}+3)$ will stand for $C(1^{x,4}+2^{r,0}+3^{0})$. A convenient consequence of this notational abuse is that κ-proteins become a particular case of $m\kappa$-agents.

Solutions are built as in Section 3. The "new" operator is now binding both kinds of names, $\mathcal{E}$ and $\mathcal{G}$, and the accompanying notion of free names is extended to include group names. Structural congruence is unchanged and in particular scope extrusion $\mathsf{S},(x)(\mathsf{S}') \equiv (x)(\mathsf{S},\mathsf{S}')$ applies both for x in $\mathcal{E}$ and $\mathcal{G}$, with the usual side-condition that $x \notin \mathrm{fn}(\mathsf{S})$.

The logs, that is the additional information on sites, can be forgotten by means of the following *projection* map:

$$(\imath^{x,n})^{-} = \imath^{x} \quad (\imath^{r,n})^{-} = (\imath^{v,n})^{-} = \imath^{v} \quad (\imath^{h,n})^{-} = \imath^{h}$$

This projection extends in the obvious way to interfaces, agents and solutions.

5.2. Interactions

In $m\kappa$-calculus *at most two agents* may interact at a time.

Definition 7. Let L, R be two pre-solutions, $\mathsf{L} \to \mathsf{R}$ is said to be an *interaction* if:
- both L and R consist of at most two agents,
- $\mathrm{fn}(\mathsf{L}) \supseteq \mathrm{fn}(\mathsf{R})$,
- L does not contain any "new" on group names.

No specific condition is demanded for the group names, except that free names of the right hand side also occur in the left hand side, and that the left hand side does not restrict them. This latter technical proviso is there only to ensure that interactions can be translated in π-calculus. Since group names have no specific constraint such as graph-likeness, it is very unlikely that one can express in π-calculus an interaction such as $(r)(A(1^{r})) \to A(1)$ which amounts to testing whether one is the only agent knowing a name in a solution.

Definition 8. An $m\kappa$-interaction $\mathsf{L} \to \mathsf{R}$ is said to be *monotonic* (resp. *antimonotonic*) if its projection $(\mathsf{L})^{-} \to (\mathsf{R})^{-}$ is a monotonic (resp. antimonotonic) κ-reaction.

General interactions, as defined above, could also be translated to π. But, one is really concerned in this study with monotonic or antimonotonic interactions, and actually a restricted class of them, which are used to translate κ in $m\kappa$. So we suppose thereafter that all interactions are monotonic or antimonotonic, and consider only $m\kappa$-solutions which are graph-like in the specific sense that they project to graph-like κ-solutions.

Recall from Section 3 that these graph-like solutions are stable under monotonic reactions, so that these assumptions are sensible.

Here are two examples of interactions:

$$A(1^x + 2), B(1^r + 2^x) \to A(1^{x,1} + 2^r), B(1^r + 2^{x,1})$$
$$A(1 + 2^r), B(1 + 2^r + 3^{r,2}) \to (x)(A(1^x + 2^r), B(1^x + 2^{r,1} + 3^{r,2})).$$

The first interaction is both monotonic and antimonotonic, only logs and states names are changed, and therefore it projects to an identical κ-reaction.

Since the group names in $\mathcal{G}$ are forgotten by the projection in κ, reactants in monotonic and antimonotonic interactions have to be graph-like only with respect to edge names in $\mathcal{E}$. The second reaction is monotonic although $r \in \mathcal{G}$ occurs more than twice on the right. In the next section, group names will be used to distinguish concurrent instances of a reaction, so not for representing edges.

Renamings have to respect the two kinds of names, that is they have to send $\mathcal{E}$ to $\mathcal{E}$ and $\mathcal{G}$ to $\mathcal{G}$. That said, the notions of matching and transition system extend easily.

6. From κ-calculus to $m\kappa$-calculus

To decompose a κ-reaction in the $m\kappa$-calculus, we follow a protocol that gradually recruits reactants and constructs the products by means of only binary and unary interactions. This protocol consists of a first phase of *recruitment* and a subsequent phase of *completion*.

Recruitment begins with a signal sent by a specific agent called the *initiator*. Then one sends and propagates two kinds of signals: *downward* signals to recruit the necessary reactants, and *upward* signals to report success back to the initiator. At the end of this first phase, the initiator knows that the global κ-reaction can be completed, and in the completion phase, this information is propagated to the other reactants.

To ship the various signals around, we need some statically predetermined structure which we now define together with some useful notation pertaining to the ways in which these signals may or may not propagate.

6.1. Scenarios

Definition 9 (Micro-scenario). Let $\mathfrak{r} = \mathsf{L} \to (\tilde{x})\mathsf{R}$ be a monotonic κ-reaction, a *micro-scenario* for $\mathfrak{r}$ is a triple $(\mathcal{F}_\mathfrak{r}, \mathcal{T}_\mathfrak{r}, \mathrm{init})$ such that:
– $\mathcal{F}_\mathfrak{r}$ is (isomorphic to) an acyclic orientation of $[\![\mathsf{R}]\!]_g$, called a *flow graph*;
– $\mathcal{T}_\mathfrak{r}$ is a tree spanning $\mathcal{F}_\mathfrak{r}$;
– init, also written $\mathrm{init}(\mathcal{F}_\mathfrak{r})$, is the common root of $\mathcal{F}_\mathfrak{r}$ and $\mathcal{T}_\mathfrak{r}$;
– and init belongs to $[\![\mathsf{L}]\!]_g$ (up to the isomorphism above) if $\mathsf{L} \neq \mathsf{0}$.

We recall that for a solution S, $[\![\mathsf{S}]\!]_g$ denotes the associated graph-with-sites (the $[\![.]\!]_g$ notation was defined in Section 3).

Without loss of generality, we assume that $\mathcal{F}_\mathfrak{r}$ is an oriented graph-with-sites with *integers* as nodes and that scenarios for different κ-reactions are chosen so as to use

disjoint set of nodes. This way agents will identify which global κ-reaction they take part in, and what role they have in it, as soon as they are recruited and handed a node of $\mathcal{F}_\tau$.

Such micro-scenarios always exist. Any connected graph admits an acyclic orientation which can be obtained, for instance, by choosing an arbitrary root, constructing a depth-first tree spanning the graph, and directing all remaining edges according to the tree ordering [9]. Thus, $[\![\mathsf{R}]\!]_g$ being connected by monotonicity, there always is a micro-scenario for any given κ-reaction: any node of $[\![\mathsf{L}]\!]_g$ can be chosen as the root and one could even assume $\mathcal{T}_\tau$ to be depth-first.

There could be loops in $[\![\mathsf{R}]\!]_g$, that is edges from a node to itself. So to be completely precise, one should say that $\mathcal{F}_\tau$ is an orientation $[\![\mathsf{R}]\!]_g$ which is acyclic except for these loops. To escape notational trouble, one may as well suppose that $[\![\mathsf{R}]\!]_g$ does not have loops. The techniques described here adapt very easily to this case, since loops are purely local to a node.

One can think of $\mathcal{F}_\tau$ as a map over sites and write accordingly:

$$\begin{array}{ll} \mathcal{F}_\tau(a,i) = b,j & \text{if } (a,i),(b,j) \text{ are connected in } \mathcal{F}_\tau, \\ \mathcal{F}_\tau(a,i) = \bot & \text{if } a,i \text{ is free in } \mathcal{F}_\tau. \end{array}$$

Considered as a map, $\mathcal{F}_\tau$ is a partial involution, and we write $\mathcal{F}_\tau^\star$ for its inverse which, of course, corresponds to the reverse orientation of the underlying undirected graph $[\![\mathsf{R}]\!]_g$. This inverse $\mathcal{F}_\tau^\star$ is a scenario only in the special case when $\mathcal{F}_\tau$ has only one sink. The same considerations apply to $\mathcal{T}_\tau$ and we will also use the map notation in this case. One may decompose $\mathcal{F}_\tau$ uniquely as $\mathcal{T}_\tau + \mathcal{T}_\tau^c$. We will use this notation later for the complement of $\mathcal{T}_\tau$.

Since $\mathcal{F}_\tau$ is an involution, which has no fixed points (no site can bind to itself), bounded sites in $[\![\mathsf{R}]\!]_g$ are naturally partitioned between *input* and *output* sites. A site is an output if it belongs to the domain of $\mathcal{F}_\tau$, and an input if it belongs to its range. In other words, a site (a,i) is an output if $\mathcal{F}_\tau(a,i) \neq \bot$ and an input if $\mathcal{F}_\tau^\star(a,i) \neq \bot$.

Definition 10 (signal ordering). Define a binary relation over sites, written $\succ$, as the smallest transitive relation such that:

$$\begin{array}{ll} \mathcal{F}_\tau(a,i) = (b,j) & \Rightarrow (a,i) \succ (b,j) \\ (a,i) \text{ input}, (a,j) \text{ output} & \Rightarrow (a,i) \succ (a,j) \end{array}$$

Since $\mathcal{F}_\tau$ is acyclic, $\prec$ is a (strict) finite partial order on sites. The same order can be defined from $\mathcal{T}_\tau$, and they coincide if and only if $\mathcal{T}_\tau$ is depth-first.

Definition 11 (input/output interfaces). For $n \in \mathbb{N}$, $a \in \mathcal{F}_\tau$ and $\tilde{x}$ a tuple indexed over the set of inputs (resp. outputs) of a and with values in $\mathcal{E}$, we define the *input* (resp. *output*) interfaces:

$$\begin{aligned} \mathrm{IN}_a^{\tilde{x},n} &:= \sum_{\{i \mid \mathcal{F}_\tau^\star(a,i) \neq \bot\}} i^{x_i,n}, \\ \mathrm{OUT}_a^{\tilde{x},n} &:= \sum_{\{i \mid \mathcal{F}_\tau(a,i) \neq \bot\}} i^{x_i,n}. \end{aligned}$$

Table 5
Initiation and first contacts

$$\frac{a = \mathrm{init}(\mathcal{F}_{\mathfrak{r}})}{A(\sigma) \to (r)(A^{r,a}(\sigma'))}\ \mathsf{init}$$

$$\frac{\mathcal{T}_{\mathfrak{r}}(a,i) = (b,j),\ x \in \mathrm{fn}(\mathfrak{r})}{A^{r,a}(\mathrm{IN}_a^{\tilde{y},1} + i^x), B(j^x + \sigma) \to A^{r,a}(\mathrm{IN}_a^{\tilde{y},1} + i^{x,1}), B^{r,b}(j^{x,1} + \sigma')}\ \mathsf{FC}_1$$

$$\frac{\mathcal{T}_{\mathfrak{r}}(a,i) = (b,j),\ x \notin \mathrm{fn}(\mathfrak{r}),\ b \in \mathsf{L}}{A^{r,a}(\mathrm{IN}_a^{\tilde{y},1} + i), B(j + \sigma) \to (x)(A^{r,a}(\mathrm{IN}_a^{\tilde{y},1} + i^{x,1}), B^{r,b}(j^{x,1} + \sigma'))}\ \mathsf{FC}_2$$

$$\frac{\mathcal{T}_{\mathfrak{r}}(a,i) = (b,j),\ x \notin \mathrm{fn}(\mathfrak{r}),\ b \notin \mathsf{L}}{A^{r,a}(\mathrm{IN}_a^{\tilde{y},1} + i) \to (x)(A^{r,a}(\mathrm{IN}_a^{\tilde{y},1} + i^{x,1}), B^{r,b}(j^{x,1} + \sigma))}\ \mathsf{FC}_3$$

These interfaces describe states of a where all inputs (resp. outputs) have the same log; they therefore have disjoint domains, and their union is also a partial interface for a.

6.2. The monotonic protocol

A protein with name A is translated as an agent of the same name but with one more *auxiliary* site, written $*$:

$$[\![A(\rho)]\!]_m = A(* + \rho).$$

That special site $*$ is used to log what little additional information one needs, that is the role of A in a given reaction (that is which node it corresponds to in $\mathcal{F}_{\mathfrak{r}}$) and a group name identifying uniquely the current attempted high-level reaction. This translation extends to κ-solutions and likewise, if S is a κ-solution, we write $[\![\mathsf{S}]\!]_m$ to denote its translation.

The purpose of this subsection is now to extend this translation to κ-reactions: given a κ-reaction $\mathfrak{r}$, one wants to define an associated family, $[\![\mathfrak{r}]\!]_m$, of $m\kappa$-interactions capable of simulating $\mathfrak{r}$ in a sense that will be made precise below. This family depends on the choice of a micro-scenario for $\mathfrak{r}$. By no means is there a unique solution to the self-assembly, and we could do with more than one initiator (as in [8]), without a spanning tree, etc. Even in the restricted kind of micro-scenarios that we are considering, there is room for different choices.

That said, we suppose now that a choice of a micro-scenario has been made and proceed to the definition of $[\![\mathfrak{r}]\!]_m$. Such a definition will depend on whether the reaction of interest is monotonic or antimonotonic. We give below a detailed account of the monotonic case, and only sketch the antimonotonic case which is much simpler. The next subsection will discuss further the antimonotonic case and various properties of the translation, while the last one sketches a correctness argument.

Interactions in $[\![\mathfrak{r}]\!]_m$ are divided into recruitments shown in Tables 5 and 6, and completions shown in Table 7. To ease reading we have systematically abbreviated

Table 6
Later contacts and responses

$$\frac{\mathcal{T}^c_{\mathfrak{r}}(a,i)=(b,j),\ x\in \mathrm{fn}(\mathfrak{r})}{A^{r,a}(\mathrm{IN}^{\tilde{y},1}_a+i^x),B^{r,b}(j^x)\to A^{r,a}(\mathrm{IN}^{\tilde{y},1}_a+i^{x,1}),B^{r,b}(j^{x,1})}\ \mathsf{LC}_1$$

$$\frac{\mathcal{T}^c_{\mathfrak{r}}(a,i)=(b,j),\ x\notin \mathrm{fn}(\mathfrak{r})}{A^{r,a}(\mathrm{IN}^{\tilde{y},1}_a+i),B^{r,b}(j)\to (x)(A^{r,a}(\mathrm{IN}^{\tilde{y},1}_a+i^{x,1}),B^{r,b}(j^{x,1}))}\ \mathsf{LC}_2$$

$$\frac{\mathcal{F}_{\mathfrak{r}}(a,i)=(b,j)}{A^{r,a}(i^{x,1}),B^{r,b}(j^{x,1}+\mathrm{OUT}^{\tilde{y},2}_b)\to A^{r,a}(i^{x,2}),B^{r,b}(j^{x,2}+\mathrm{OUT}^{\tilde{y},2}_b)}\ \mathsf{R}$$

Table 7
Completions

$$\frac{a=\mathrm{init}(\mathcal{F}_{\mathfrak{r}}),}{A^{r,a}(\mathrm{OUT}^{\tilde{y},2}_a)\to A^{r,a}(\mathrm{OUT}^{\tilde{y},3}_a)}\ \mathsf{shift}$$

$$\frac{a=\mathrm{init}(\mathcal{F}_{\mathfrak{r}}),\ \mathcal{F}_{\mathfrak{r}}(a,i)=(b,j)}{A^{r,a}(i^{x,3}),B^{r,b}(j^{x,2})\to A^{r,a}(i^{x,4}),B^{r,b}(j^{x,3})}\ \mathsf{i\text{-}ppg}$$

$$\frac{a\neq\mathrm{init}(\mathcal{F}_{\mathfrak{r}}),\ \mathcal{F}_{\mathfrak{r}}(a,i)=(b,j)}{A^{r,a}(\mathrm{IN}^{\tilde{y},3}_a+i^{x,2}),B^{r,b}(j^{x,2})\to A^{r,a}(\mathrm{IN}^{\tilde{y},3}_a+i^{x,3}),B^{r,b}(j^{x,3})}\ \mathsf{ppg}$$

$$\frac{a=\mathrm{init}(\mathcal{F}_{\mathfrak{r}})}{A^{r,a}(\mathrm{OUT}^{\tilde{x},4}_a)\to A(o^{\tilde{x}}_a)}\ \mathsf{i\text{-}exit}$$

$$\frac{a\neq\mathrm{init}(\mathcal{F}_{\mathfrak{r}})}{A^{r,a}(\mathrm{IN}^{\tilde{y},3}_a+\mathrm{OUT}^{\tilde{z},3}_a)\to A(\iota^{\tilde{y}}_a+o^{\tilde{z}}_a)}\ \mathsf{exit}$$

$A(*^{r,a}+\theta)$ as $A^{r,a}(\theta)$ and also made use of the notation for input and output interfaces introduced above. On top of the inference rule the defining conditions for the interaction are given.

A perfunctory glance at these tables shows that all interactions involve at most two agents, as recommended in Definition 7, that at most one edge is created at a time and that the option of using reactants and products which are not strongly graph-like is used a lot. We let the reader verify that the rest of the conditions are satisfied as well, and that these interactions, which we are going now to review closely, are in fact all monotonic in the sense of Definition 8.

Initiation (*init*). The *initial* reaction is possible for any quiescent agent bearing the correct name, it is the moment when the name r for the current attempt of a reaction is created. It is understood in the notation, taken here and in the sequel, that a and b have respective names A and B in the flow graph.

Take note that the name map from nodes to names in $\mathcal{P}$ is not injective. In the limit case, not unheard of in biology, there could be a complicated reaction involving only copies of one and the same protein. This is indeed why roles are needed.[4]

During initiation, one also verifies and modifies the initiator free interface. Specifically, σ and σ' have as common domain the free sites of init in $\mathcal{F}_\tau$, and respectively map these to the values specified by L and R. Take note that only sites which are free in R (hence free in L as well, by monotonicity) are tested and perhaps modified at this stage. Sites which are free in L and bound in R (hence all visible in L) are taken care of in steps FC_2 and LC_2 (see below).

First contacts (FC_1, FC_2, FC_3). The next three interactions are the *first contacts*. If $\mathcal{T}_\tau(a,i)=(b,j)$, then there are three forms of first contact which we can list here in increasing order of creativity:

– the edge $(a,i;b,j)$ may already exist in L;
– it may have to be created while the other end exists in L;
– finally both the edge and the other end may have to be created.

In all three cases, the newly contacted agent, named B, is handed a role b, which it logs on its special $*$ site, and the contact is logged with a 1 on both sides.[5]

As in the initiation case, first contacts are the occasion to verify and modify the free interface of the newly contacted agent. In the case of FC_1 and FC_2, σ and σ' have as common domain the free sites of b in $\mathcal{F}_\tau$, and respectively map these to the values specified by L and R. The case of node creation, FC_3, is different. One takes σ to be the interface defined on $\mathfrak{s}(B)\backslash\{j\}$ mapping sites bound in R to v (to allow for later binding) and sites free in R to the values specified by R.

Later contacts (LC_1, LC_2). Then come the two *later contacts* (not needed when $\mathcal{F}_\tau$ is a tree). Indeed, if $\mathcal{F}_\tau(a,i)=(b,j)$ and $\mathcal{T}_\tau(a,i)=\perp$, there can only be two forms of contacts, since the other end of the edge must have been contacted already, which means in particular that it already exists. Upon a later contact one does not need to check the free interfaces, neither does one need to pass over a role, because this has been taken care of by the first contact, and it is enough to just log the contact, again with a 1 on both sides.

A noteworthy point is that each contact, first or later, is demanding that the *inputs* of the contacting agent have log 1. This imposes a constraint on the way signals may propagate which we will discuss later. Let us just say for now that this constraint would be too strong if the flow graph were not acyclic.

A remark of lesser importance is that this set of contact interactions behaves correctly also in the case where $\mathcal{F}_\tau$ has parallel edges (edges with the same source and target).

[4] This important fact was overlooked in the short version [8] of the present paper, resulting in a self-assembly which was only correct when all participants in the reaction had statically different names. Roles allow to handle the general case by dynamically allocating different names for all reactants. First contacts are "logged" and therefore are taken at most once for each reaction attempt. As a consequence, the allocation of roles is injective over the set of agents participating in a same attempt.

[5] The tree $\mathcal{T}_\tau$ spans $\mathcal{F}_\tau$, and therefore it selects among all predecessors of a given b different from init, its *parent*, that is the only a such that $\mathcal{T}_\tau(a,i)=(b,j)$ for some i and j. This parent is responsible here, in the recruitment phase, for contacting first (some agent of name) B and hand him over the role b. We will show later an example of what kind of failure can happen when all contacts concur freely.

At most one of any group of parallel edges will belong to the spanning tree and will be dealt with in a first contact interaction, the others will be dealt with by a later contact. No special workaround is needed here.

Response (R). Finally the last type of interaction in the recruitment phase is the *response* which pushes a signal upwards back to the initiator. This is only when all successors in $\mathcal{F}_\tau$ have already responded. Such a step can be freely taken by leaves of $\mathcal{T}_\tau$, once they are contacted, since leaves have no successors and therefore the condition on the logs of their output interface, is vacuously satisfied. Again it is important that the flow graph be acyclic, else one would have no chance to fire any of these responses.

Phase shift (shift). The second phase begins with the *phase-shift* interaction. At that very moment, the initiator knows that everything has gone well, and the right hand side R has been correctly built.

Propagation and Exit (i-ppg, ppg, i-exit, exit). The global reaction is over and it is enough to complete the process by sending down a success signal, materialized by the log 3, which will let all other agents recruited in the reaction know that the reaction has succeeded. So the signal is sent downwards to the leaves and when an agent has received the signal at *all* inputs and passed it at *all* outputs, it finally may exit asynchronously and project again to a quiescent κ-like agent.

Because of the assymmetry between the initiator and the other agents, one needs two different propagation and exit rules, but they are really of the same kind.

6.3. Discussion

Antimonotonic reactions have a connected left hand side and therefore the corresponding recruitment is much simpler. It consists of checking that the neighborhood of the initiator indeed contains the left hand side. Since nothing is created, node or edge, one only needs init, FC_1, LC_1 and R, before the phase shift, all of which project to an identity reaction in κ. Dually to monotonic reactions, all deletions, will be performed after the phase shift, if one reaches it. This is easily implemented using the same flow graph and we do not give the explicit interactions. Yet, we do take note that, in antimonotonic reactions, no actual change is done this side of the phase shift to the underlying κ-solution.

From this discussion one also sees that composite reactions, that is to say the reactions which one can decompose as a monotonic reaction followed by an antimonotonic one (which we discussed earlier at the end of Section 3.4), can easily be implemented as well. This side of the phase shift one does the monotonic part of the job, and beyond one does the rest. This substantiates the claim that one could take them as basic as well: they are as easy to deal with in $m\kappa$ as are the monotonic and antimonotonic ones.

The spanning tree. We have already observed that the spanning tree serves as a way of imposing a "parental priority" between the contacts. Only the parent according to $\mathcal{T}_\tau$ is allowed to wake a child and recruit it, while all the other reactants have to use a further contact interaction.

If we do not do this, some strange self-deadlocks may happen. For instance, starting with the following κ-reaction and solution:

$$\begin{aligned} \mathsf{t} &= A(1^x + 2), B(1^x + 2), C(1 + 2) \to \\ &\quad (yz)(A(1^x + 2^y), B(1^x + 2^z), C(1^y + 2^z)) \\ \mathsf{T} &= (xyz)(A(1^x + 2), B(1^x + 2), C(1 + 2), C(1 + 2)) \end{aligned}$$

and supposing there is no priority between contacts, then A and B might recruit distinct Cs in T, and so, in some sense, the recruitment defeats itself all alone. Having the tree to sort out who is doing the first contact, and who is not, solves the question.

Depth-first. Suppose now the spanning tree is depth-first, and some agent A is contacted for the first time, then A's predecessors in $\mathcal{F}_\tau$ are A's predecessors in $\mathcal{T}_\tau$, hence they are already recruited and connected to A. Therefore, if we follow the intuition that connected communication is *instantaneous*, because it is the symbolic counterpart of interaction between components that were already brought in physical contact, then the only interactions where some waiting is needed, and maybe some timeout if one thinks about a time-conscious implementation, are of the FC_2 kind. They correspond biologically to collisions in the solution, a process which indeed takes time.

Collisions. By the way, we take note that the only non-deterministic steps in our collection of interactions are precisely of the FC_2 and init kind. All the others are deterministic in the sense that given the interaction and any one of the reacting agents, there can be at most one other agent such that the pair matches the interaction. These are also the only interactions where the blind search mechanism embodied in the interactions can fail.

Observable interactions. To conclude this series of remarks on the decomposition, we note that few interactions are actually observable in the higher language: FC_2, FC_3, LC_2 where all the creative work is done and init and FC_1 where free interfaces might be modified. All other interactions project to identical κ-reactions, they are only modifying the logs and, semantically, only serve the purpose of propagating information.

6.4. Simulation and correctness

Thereafter and for the rest of the section, we suppose we deal with an $m\kappa$ system obtained by the translation we just defined. We will still write $\to^*$ for both associated transition systems. The notions we manipulate below make no sense in general in $m\kappa$ and are only meaningful for those particular translated systems.

An important invariant that our set of interactions respects is that, at any moment, the set of agents bearing a given group name r will be connected. These connected sub-solutions are all disjoint, because initial reactions generate a new group name that uniquely identifies the "session" one begins, and upon first contact one commits oneself to only participate in the current session (see interactions FC_1, FC_2 and FC_3) and thereafter only interact with agents in the same session (see all other binary interactions). We logically call these disjoint connected set of agents *groups*. Agents not belonging to any group were called *quiescent*.

Each group involves agents currently attempting some instance of a κ-reaction. This reaction may or may not succeed. We say a group is monotonic or antimonotonic, depending on whether the associated κ-reaction one is trying to complete is a monotonic or an antimonotonic one.

Whenever a high-level reaction has a match in a solution, then the firing of the reaction can be simulated in the corresponding low-level solution.

Proposition 4. *Let* S, T *be* κ*-solutions*: *if* $\mathsf{S} \to^* \mathsf{T}$ *then* $[\![\mathsf{S}]\!]_m \to^* [\![\mathsf{T}]\!]_m$.

Proof (Sketch). It is enough to prove it for a one-step transition, that is for the application of some κ-reaction $\mathfrak{r} = \mathsf{L} \to (\tilde{x})(\mathsf{R})$. Since S can make a one-step transition to T, there must be a corresponding κ-matching, and therefore R is, up to renaming and extension, a sub-solution in T. By composing this inclusion with the isomorphism between $\mathcal{F}_{\mathfrak{r}}$ and $[\![\mathsf{R}]\!]_g$ we find who is the initiator in the reaction and fire the init rule with it. This creates a unique identifier for the attempted reaction, say r. Now, it is up to us to define the order in which to fire all the interactions, since the correctness statement demands nothing more.

A possibility is to grow R by using only contact interactions to begin with. There are only two things to take care of. First, for a non-deterministic contact FC_2 between A and B, one has to use again the embedding above to guess which agent B, agent A has to create an edge to. Second, one has to contact sites not just in any order, but in a way consistent with the constraints on inputs asked for in the contact interactions. These constraints amount to asking that at any time the r-group be upward-closed with respect to the signal ordering $\prec$ (see Definition 10). Such a growth constraint is always satisfiable because of acyclicity (actually it is satisfiable if and only if the flow graph is acyclic). This somewhat subtle point can be proved either directly, or by using a depth-first $\mathcal{T}_{\mathfrak{r}}$. In the latter case we already noticed that all ancestors of an agent are recruited before the agent itself is, so that an upward-closed growth is obtained by always performing all later contacts LC_1, LC_2, between an agent and its ancestors immediately after recruiting him. (One sees well that these constraints only operate when the flow graph is not a tree.)

Once the contacts are all done, all edges have both their ends with log 1, and it remains to move up the response back to the initiator by using the response interaction. This time one has to find a downward-closed way to proceed, which is possible for the same reason.

When the various signals have reached the initiator, the recruitment is over, and one can trigger the phase shift and the subsequent interactions. □

We observe that each high-level step generates a number of small steps which is $3e_{\mathfrak{r}} + n_{\mathfrak{r}} + 1$, where $e_{\mathfrak{r}}$ is the number of edges in the right hand side R of $\mathfrak{r}$, and $n_{\mathfrak{r}}$ the number of nodes. So the simulation stays linear in the size of R.

This first result says very little in terms of correctness and one would also want to know that the low-level $m\kappa$-system does not generate any solutions the projections of which would be unreachable from the higher level solution.

Definition 12. Given κ-solutions S and T such that $[\![\mathsf{S}]\!]_m \to^* \mathsf{T}$, one defines the *cleanup* of T, written T^c, as the $m\kappa$-solution obtained by:
- completion of groups which are past the phase shift (see Table 7),
- projection of antimonotonic groups which are pre-phase-shift (see Section 5.1),
- deletions in the monotonic groups which are pre-phase-shift (see discussion below), and
- erasure of all the auxiliary $*$ sites.

Completions are always possible because past the phase shift, one cannot fail. Antimonotonic groups only have to be projected, since antimonotonic recruitment is pure checking and nothing changes except the logs. Finally, in monotonic groups one has to delete any edges and proteins that were added in the creative steps FC_2, FC_3 and LC_2 (the only ones with a "new" on the products side).

The correctness of our solution of the self-assembly question can then be stated as follows.

Theorem 5. *Let* S *be a* κ*-solution*: *if* $[\![\mathsf{S}]\!]_m \to^* \mathsf{T}$ *then* $\mathsf{S} \to^* \mathsf{T}^c$.

Proof (Sketch). One proves first by induction on all possible interactions that this side of the phase shift, a group has its sites with log 1 (resp. 2) forming an $\prec$-upward-closed (resp. $\prec$-downward-closed) connected subset of all the group sites. An immediate consequence is that, when the phase shift happens (just before it happens, to be precise), all the logs of the group bounded sites are set at 2, and the group is, up to renaming and extensions, equal to R. Indeed, the only downward-closed subset of sites of $\mathcal{F}_{\mathsf{t}}$ that contains all the initiator sites, is the set of *all* sites. This is a consequence of the definition of $\prec$. Now the response interactions are pairing the sites together exactly as they are in $\mathcal{F}_{\mathsf{t}}$. And all responses have happened, else some logs would still be set strictly below 2. Therefore, a reaction attempt succeeds if and only if it reaches the phase shift, and at that point it is always completable. If it does not reach the phase shift, it can be cleaned up as explained above. □

We have seen that the decomposition is a simulation and never makes any mistakes. It is now time to discuss in which sense the decomposition can fail, or in other words what kind of further notion of correctness one could try to reach for.

A first case when deadlock may happen is when κ-reactions are competing with another on the same reactants. In this case, the conflicting reactions, which can be occurrences of the same reaction, can be initiated concurrently in $m\kappa$ and run out of resources resulting in a deadlock.

A second case of deadlock is when a single decomposed κ-reaction gets stuck in its search space in $m\kappa$. Imagine for instance the following κ-solution and κ-reaction:

$$\begin{aligned}\mathsf{S} &= A(1+2), B(1+2^{x_1}), C_1(1^{x_1}), B(1+2^{x_2}), C_2(1^{x_2})\\ \mathfrak{s} &= A(1+2), B(1+2^{x_1}), C_1(1^{x_1}), B(1+2^{x_2}), C_2(1^{x_2}) \to\\ &\quad (y_1 y_2)(A(1^{y_1}+2^{y_2}), B(1^{y_1}+2^{x_1}), C_1(1^{x_1}), B(1^{y_2}+2^{x_2}), C_2(1^{x_2})).\end{aligned}$$

According to the translation, and in essence, A has to guess, with two instances of FC_2, which B it has to bind to, on its first site, and which on its second. Since both Bs look

exactly the same as far as A can tell in a *local* interaction, our series of interactions can very well try to produce the twisted complex:

$$A(1^{y_2}+2^{y_1}), B(1^{y_1}+2^{x_1}), C_1(1^{x_1}), B(1^{y_2}+2^{x_2}), C_2(1^{x_2}).$$

Suppose A was the initiator, and it guessed wrongly giving role b_1 to the B connected with C_2 and b_2 to the other B, then, the group is stuck since the newly recruited Bs will try to fire the following FC_1 interactions:

$$B^{r,b_1}(1^{y,1}+2^{x}), C_1(1^{x}) \to B^{r,b_1}(1^{y,1}+2^{x}), C_1^{r,c_1}(1^{x})$$
$$B^{r,b_2}(1^{y,1}+2^{x}), C_2(1^{x}) \to B^{r,b_2}(1^{y,1}+2^{x}), C_2^{r,c_2}(1^{x})$$

and none will succeed, since B^{r,b_1} is connected to C_2 and conversely. The mistake does not spread further since both agents B can see they are locally not connected with the C_i they should be connected with, according to the role that A has bestowed upon them.

To summarize, one has two sources of deadlocks, contentions and guesses. Thus, one can think of extending $[\![\mathfrak{r}]\!]_m$ by adding some pre-phase-shift interactions that will invert the contact and response interactions and give to a group the ability to escape such deadlocks. This seems reasonable as long as agents can test whether they can take a step backwards without inconsistencies. The invariant to preserve here is downward (resp. upward) closure for the sites with log 2 (resp. 1).

Consider the simpler example of the response interaction R, where there is nothing to reverse except the signal itself. We recall the plain forward version and write just below the backward dual one:

$$A^{r,a}(i^{x,1}), B^{r,b}(j^{x,1}+\mathrm{OUT}_b^{\tilde{y},2}) \to A^{r,a}(i^{x,2}), B^{r,b}(j^{x,2}+\mathrm{OUT}_b^{\tilde{y},2})$$
$$A^{r,a}(\mathrm{IN}_a^{\tilde{y},1}+i^{x,2}), B^{r,b}(j^{x,2}) \to A^{r,a}(\mathrm{IN}_a^{\tilde{y},1}+i^{x,1}), B^{r,b}(j^{x,1})$$

Instead of verifying that all outputs to B are set at 2, one now verifies dually that no input of A has been set at 2 yet, i.e., the signal has not gone further. This way the key closure invariants are clearly preserved.

Smarter agents and smarter management of failures, for instance with timeouts, alarms and alarm propagation, or more brutish management with checkpoints and back-ups, might be interesting in a richer time-conscious framework, e.g., in robotics [17], distributed system design, or transaction models [2]. Anyway, it still remains to be seen if one can write down a correct set of reversed interactions and we have not checked all the details. Such a set would somehow internalize the cleanup procedure described above, and escape deadlocks, while keeping the agent reasonably dumb as suits a formal biological model.

7. From $m\kappa$-calculus to π-calculus

In this final section we begin with a brief introduction to π-calculus [20], and then detail the compilation of $m\kappa$-calculus. Our compilation is "protein-centric", that is to say proteins are translated as processes whose behavior is obtained from all the interactions

they participate to. This is in line both with Regev's direct representations of various biological pathways [23,26], and with the second encoding provided in a preceding paper dealing with a multiset-based version of κ [7].

7.1. The π-calculus

The π-calculus uses three countable sets:

- *names*, $\mathcal{N}$, ranged over by x, y, z,...
- *agent names*, ranged over by $\mathtt{A}$, $\mathtt{B}$,...
- *variables* X, Y, Z,...

As usual, we address tuples with $\tilde{u}$ and write $\{\tilde{u}\}$ for the corresponding set. We use a π-calculus extended with natural numbers where we distinguish three syntactic categories, *values* written u, *matches* written M, and *processes* written P. The grammar is detailed below.

$$\begin{array}{lll} u & := n \mid x \mid X & \text{values} \\ M & := [u=u] \mid M\,M & \text{matches} \\ P & := \mathbf{0} \mid \tau.P \mid x(\tilde{X}).P \mid \bar{x}\langle\tilde{u}\rangle.P \mid P+P & \text{processes} \\ & \quad P \mid P \mid M\,P;P \mid (x)P \mid \mathtt{A}(\tilde{X}) & \end{array}$$

Values are natural numbers, names, and variables. Variables represent formal parameters, and sometimes, when the corresponding parameter is a name, we simply use a name rather than a variable. Matches are sequences of equalities between values. A process can be the inert process $\mathbf{0}$, a process performing an internal move, an input $x(\tilde{X}).P$, an output $\bar{x}\langle\tilde{u}\rangle.P$, a choice, a parallel composition, a match guarding two processes, a restricted process of the form $(x)P$ where (x) is the "new" operator that limits the scope of x to P, or an agent invocation $\mathtt{A}(\tilde{u})$, in which case we ask for a unique equation $\mathtt{A}(\tilde{X}) := P$ defining $\mathtt{A}$. Restrictions bind names, that is (x) in $(x)P$ binds the name x wherever it is free in P and likewise, input and agent definition bind variables, that is $x(\tilde{X}).P$ and $\mathtt{A}(\tilde{X}) := P$ bind the free occurrences of the variables $\tilde{X}$ in P. Names and variables that are not bound are called *free* as usual and we write $\mathrm{fn}(P)$ for the set of such names and variables in P as we did in κ.

Table 8 collects the semantics of π-calculus, except for the symmetric forms of rules (SUM) and (PAR) which are omitted. The semantics is described as a transition system on syntactic processes with transitions labelled by certain *actions*. As can be inferred from Table 8, actions, written μ, are of three types: *internal* actions τ, *inputs* $x(\tilde{Y})$ and *outputs* $(\tilde{y})\bar{x}\langle\tilde{u}\rangle$. When $(\tilde{y})$ is not empty in an output action, one says it is a *bounded output*. This tuple of names $\tilde{y}$ represents the bounded names that the process is exporting to the context. Bounded outputs, $(\tilde{y})\bar{x}\langle\tilde{u}\rangle$, generated by the transitions above all satisfy: (1) $\tilde{y}\subseteq\tilde{u}$ and (2) $x\notin\tilde{y}$. One defines:

$$\begin{array}{llll} \mathrm{fn}(\tau) & = \varnothing & \mathrm{bn}(\tau) & = \varnothing \\ \mathrm{fn}(x(\tilde{Y})) & = \{x\} & \mathrm{bn}(x(\tilde{Y})) & = \{\tilde{Y}\} \\ \mathrm{fn}((\tilde{y})\bar{x}\langle\tilde{u}\rangle) & = \{x\}\cup(\mathcal{N}\cap\{\tilde{u}\}\cap\{\tilde{y}\}^c) & \mathrm{bn}((\tilde{y})\bar{x}\langle\tilde{u}\rangle) & = \{\tilde{y}\} \end{array}$$

Rules (PAR), (COM), (NEW) and (OPEN) all have side-conditions controlling bounded output and involving $\mathrm{fn}(\mu)$ and $\mathrm{bn}(\mu)$. Specifically, these conditions ensure that (1)

Table 8
Operational semantics of the π-calculus

$$\frac{}{\tau.P \xrightarrow{\tau} P}\ (\text{TAU}) \qquad \frac{}{x(\widetilde{Y}).P \xrightarrow{x(\widetilde{Y})} P}\ (\text{INP}) \qquad \frac{\text{no variable occurs in } \widetilde{u}}{\overline{x}\langle\widetilde{u}\rangle.P \xrightarrow{\overline{x}\langle\widetilde{u}\rangle} P}\ (\text{OUT})$$

$$\frac{P \xrightarrow{\mu} Q \quad x \notin \text{fn}(\mu)}{(x)P \xrightarrow{\mu} (x)Q}\ (\text{NEW}) \qquad \frac{P \xrightarrow{(\widetilde{y})\overline{x}\langle\widetilde{u}\rangle} Q \quad z \neq x \quad z \in \widetilde{u} \setminus \{\widetilde{y}\}}{(z)P \xrightarrow{(z\widetilde{y})\overline{x}\langle\widetilde{u}\rangle} Q}\ (\text{OPEN})$$

$$\frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}\ (\text{SUM}) \qquad \frac{P \xrightarrow{\mu} P' \quad \text{bn}(\mu) \cap \text{fn}(Q) = \varnothing}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ (\text{PAR})$$

$$\frac{P \xrightarrow{\mu} Q}{[u=u]P;\, R \xrightarrow{\mu} Q}\ (\text{MATCH}) \qquad \frac{R \xrightarrow{\mu} Q \quad u \neq u'}{[u=u']P;\, R \xrightarrow{\mu} Q}\ (\text{MISMATCH})$$

$$\frac{P \xrightarrow{(\widetilde{y})\overline{x}\langle\widetilde{u}\rangle} P' \quad Q \xrightarrow{x(\widetilde{Y})} Q' \quad \{\widetilde{y}\} \cap \text{fn}(Q) = \varnothing}{P \mid Q \xrightarrow{\tau} (\widetilde{y})(P' \mid Q'\{\widetilde{u}/\widetilde{Y}\})}\ (\text{COM}) \qquad \frac{P\{\widetilde{u}/\widetilde{X}\} \xrightarrow{\mu} Q}{\mathtt{A}(\widetilde{u}) \xrightarrow{\mu} Q}\ (\text{APP})$$

the exported bounded names do not capture any variables when they finally appear in the right hand side of the conclusion of rule (COM), (2) one does not send on a bound name.

These rules are standard, except for (OUT) where agent $\overline{x}\langle\tilde{u}\rangle.P$ may go to state P with a reduction labeled $\overline{x}\langle\tilde{u}\rangle$ only if it carries values that are numbers or channels. One cannot send a variable.

7.2. The translation of mκ-calculus

The translation in π-calculus, written $[\![\cdot]\!]_\pi$, is first defined on *interfaces*:

$$[\![1^{u_1,m_1} + \cdots + n^{u_n,m_n}]\!]_\pi \;=\; u_1, m_1, k_1, \ldots, u_n, m_n, k_n$$

with $k_i = 0$ if $u_i \in \{h, v\}$, $k_i = 1$ if $u_i \in \mathcal{G}$, and $k_i = 2$ if $u_i \in \mathcal{E}$.

Interfaces are encoded as tuples, the state of each site corresponding to three elements in the tuple: the state of the first site is encoded by the first three elements, that of the second with the next three, and so on.

The reader might wonder why one needs to encode a type information k_i in the state of site i. This technicality reflects the fact that the $m\kappa$ notion of matching implicitly

checks the type. For instance if

$$\mathfrak{s} = (x)(A(i^x), B(j^x)) \to A(i), B(j)$$
$$\mathsf{S} = A(i^r), B(j^r)$$

and $r \in \mathcal{G}$, $x \in \mathcal{E}$, then there is no a match between S and the $\mathfrak{s}$ left hand side. To prevent a match in the translation in π-calculus of this situation, one will dynamically typecheck the names carried by the corresponding processes $\mathtt{A}$, $\mathtt{B}$ (inserting matches of the form $[X_{3i+2} = k_i]$ and $[Y_{3j+2} = k_j]$ at suitable places, see the encoding below) and in order to do this, one has to make this type of information available to the processes.

The encoding extends naturally to *solutions*:

$$[\![A(\theta)]\!]_\pi = \mathtt{A}([\![\theta]\!]_\pi)$$
$$[\![\mathsf{S}, \mathsf{T}]\!]_\pi = [\![\mathsf{S}]\!]_\pi \mid [\![\mathsf{T}]\!]_\pi$$
$$[\![(x)(\mathsf{S})]\!]_\pi = (x)([\![\mathsf{S}]\!]_\pi)$$

where $\mathtt{A}(\tilde{X})$ is defined as

$$\mathtt{A}(\tilde{X}) := \sum_{A(\theta) \in_L \mathfrak{r} \in \mathfrak{R}_0} \mathfrak{r}_{A(\theta)}(\tilde{X}) + \sum_{A \in_R \mathbf{0} \to \mathsf{R} \in \mathfrak{R}_1} \tau.(\mathfrak{s}_\mathsf{R} \mid \mathtt{A}(\tilde{X}))$$

Notation $A(\theta) \in_L \mathfrak{r}$ means that $A(\theta)$ is a reactant in the interaction $\mathfrak{r}$. Likewise, $A \in_R \mathfrak{r}$ means that some product of $\mathfrak{r}$ has name A. The left and right sums are indexed by disjoint sets of interactions: $\mathfrak{R}_0$ is the subset of interactions in $\mathfrak{R}$ with non-empty left hand sides, while $\mathfrak{R}_1$ is the complement subset of interactions with empty left hand sides. One makes here the simplifying assumption each A participating in a $\mathfrak{R}_1$ type of interaction always occur at least once in the ambient solution.

To complete the definition of the translation, it remains to define the parameterized processes $\mathfrak{r}_{A(\theta)}(\tilde{X})$ and $\mathfrak{s}_\mathsf{R}$. In order to do this conveniently, we first set up some new notations.

7.2.1. Filters

Given a partial extended interface θ, and a tuple of variables $\tilde{X}$, $[\tilde{X} = \theta]$ denotes the following sequence of matches, for $i \in \mathrm{dom}(\theta)$:

- $[X_{3i} = h][X_{3i+1} = m][X_{3i+2} = 0]$, if $\theta(i) = h, m$;
- $[X_{3i} = v][X_{3i+1} = m][X_{3i+2} = 0]$, if $\theta(i) = v, m$;
- $[X_{3i+1} = m][X_{3i+2} = 1]$, if $\theta(i) = r, m$, and $r \in \mathcal{G}$;
- $[X_{3i+1} = m][X_{3i+2} = 2]$, if $\theta(i) = x, m$, and $x \in \mathcal{E}$;
- $[X_{3i} = X_{3j}]$, if $\theta(i) = u, m$, $\theta(j) = u, n$, and $u \in \mathcal{G} + \mathcal{E}$.

One has to suppose that $\tilde{X}$ has length greater than $3 \times \max\{\mathrm{dom}(\theta)\} + 2$,

The filter matches check whether or not a protein may participate in an interaction. In particular, the matches will verify that logs have the right value, whether a site is visible, hidden, carries a group name, or carries an edge name, and lastly (item 5) whether two occurrences of a same name in the interface are the same in the tuple.

One needs also a binary analog of this first filter operator.

Given two partial extended interfaces, θ and ψ, and two tuples of variables, $\tilde{X}$ and $\tilde{Y}$, of suitable length, $[\tilde{X}, \tilde{Y} = \theta, \psi]$ denotes the sequence of matches $[X_{3i} = Y_{3j}]$, for all $i \in \mathrm{dom}(\theta)$, $j \in \mathrm{dom}(\psi)$, such that $\theta(i) = u, m$, $\psi(j) = u, n$, and $u \in \mathcal{G} + \mathcal{E}$.

The binary filter operator produces a series of matches that cross-check whether names which are identical in θ and ψ, are also identical in $\tilde{X}$ and $\tilde{Y}$.

7.2.2. Updates

Finally, one needs to implement the effect of an interaction on an extended interface, as specified by the partial extended interfaces in its products.

Given three partial extended interfaces θ, ψ and θ', a set of names $\tilde{x}$ in $\mathcal{E}$, such that:
– (1) $\mathrm{fn}(\theta') \subseteq \mathrm{fn}(\theta) \cup \mathrm{fn}(\psi) \cup \tilde{x}$,
– (2) $\tilde{x} \cap \mathrm{fn}(\theta) = \tilde{x} \cap \mathrm{fn}(\psi) = \varnothing$,
and two tuples of variables $\tilde{X}$, $\tilde{Y}$ of appropriate lengths, one defines a tuple of *values*, $\tilde{X} \leftarrow \theta'$, of the same length as $\tilde{X}$, as follows:
– $X_{3i}, X_{3i+1}, X_{3i+2}$, if $i \notin \mathrm{dom}(\theta)$;
– $\varepsilon, m, 0$, if $\theta'(i) = \varepsilon, m$ with $\varepsilon \in \{h, v\}$;
– $x, m, 2$, if $\theta'(i) = x, m$ and $x \in \tilde{x}$;
– X_{3j}, m, τ if $\theta'(i) = u, m$, $\theta(j) = u, n$, with $\tau = 1$ (2) if $u \in \mathcal{G}$ ($\mathcal{E}$);
– Y_{3j}, m, τ if $\theta'(i) = u, m$, $\psi(j) = u, n$, with $\tau = 1$ (2) if $u \in \mathcal{G}$ ($\mathcal{E}$);

The interfaces θ, ψ represent interfaces in the left hand side of some interaction, while the interface θ' represent the interface of some agent on the right hand side, and $\tilde{x}$ stands for created edges.

This definition is a bit peculiar in that the last two clauses are ambiguous. There is not a unique tuple satisfying them, because, for instance, one could have $\theta'(i) = u, m$, $\theta(j) = u, n$ and $\theta(k)$ or $\psi(k) = u, p$. But it does not matter how one resolves this choice, since when there is ambiguity, all options lead to processes behaving identically. The notation $\tilde{X} \leftarrow \theta'$ does not mention the other needed parameters, θ, ψ, $\tilde{x}$, and $\tilde{Y}$, but these will be clear from the context.

Note also that exactly the sites that are referred to in the interface θ' of the product are modified (see first clause).

7.2.3. The processes $\mathfrak{r}_{A(\theta)}(\tilde{X})$ and $\mathfrak{s}_{\mathsf{R}}$

Below we assume that every binary monotonic interaction $\mathfrak{r}$, that is any monotonic interaction with two reactants, has a unique associated name $\upsilon_{\mathfrak{r}}$. This name represents the capacity of the agents to interact through their visible sites. We know these sites exist by definition of monotonic interactions.

Again for such binary $m\kappa$-interactions, we choose one agent to be translated as a sender, and one as a receiver. This, of course, is an artefact of the translation which has no counterpart in $m\kappa$ and only comes from the fact that the communication model of π-calculus is asymmetric. In the specific case where both reactants are *exactly* the same agent, then we give to *both* the sum of the sending and receiving behaviors.

We now enumerate all cases of interactions following the total number of agents involved, reactants and products: there are 4 agents involved in case 1–2, 3 in case 3–4, and at most 2 in case 5–8. Each time the corresponding contributions $\mathfrak{r}_{A(\theta)}(\tilde{X})$ or $\mathfrak{s}_{\mathsf{R}}$ to the behavior of the reactants is given.

Case 1: $\mathfrak{r} = A(\theta), B(\psi) \to (\tilde{x})(A(\theta'), B(\psi'))$, we take $\mathtt{A}$ to assume the role of the sender on $\upsilon_{\mathfrak{r}}$. Each agent verifies that its own current interface is compatible with the

interaction; then agent A sends two new names on the reaction channel, z for success, and z' for reset, together with its own interface represented by $\tilde{X}$; the other agent B is in charge of verifying whether A and B are properly connected and uses the matching operator to do this.

$$\begin{aligned}\mathfrak{r}_{A(\theta)}(\tilde{X}) := {} & [\tilde{X} = \theta] \\ & (zz')(\overline{\upsilon_{\mathfrak{r}}}\,\langle \tilde{X}, z, z'\rangle. \\ & (z\,(\tilde{x}, \tilde{Y}).\mathtt{A}(\tilde{X} \leftarrow \theta') + z'\,().\mathtt{A}(\tilde{X})))\end{aligned}$$

$$\begin{aligned}\mathfrak{r}_{B(\psi)}(\tilde{Y}) := {} & [\tilde{Y} = \psi] \\ & \upsilon_{\mathfrak{r}}\,(\tilde{Z}, z, z').[\tilde{Z}, \tilde{Y} = \theta, \psi] \\ & (\tilde{x})(\bar{z}\,\langle \tilde{x}, \tilde{Y}\rangle.\mathtt{B}(\tilde{Y} \leftarrow \psi'))\,;\overline{z'}\,\langle\rangle.\mathtt{B}(\tilde{Y})\end{aligned}$$

If the connection is as it should be in $\mathfrak{r}$, then B creates the new edges $\tilde{x}$ and send them on the success channel to A. B also sends its own interface, in case (group) names therein are needed to update the interface of A. Therefore both agents update their interfaces. If the connection does not match with $\mathfrak{r}$, then B sends the reset signal and both agents return to their preceding state.

Case 2: $\mathfrak{r} = (\tilde{x})(A(\theta), B(\psi)) \to A(\theta'), B(\psi')$, by definition of an antimonotonic interaction, reactants are connected, and therefore there exists i, j such that $\theta(i) = x, m$, $\psi(j) = x, n$ and $x \in \mathcal{E}$. We use this name x as a channel through which the interaction is triggered and therefore in this case there is no need of a name associated to the reaction and it is enough to send some integer $\lceil \mathfrak{r} \rceil$ coding for the reaction.

$$\begin{aligned}\mathfrak{r}_{A(\theta)}(\tilde{X}) := {} & [\tilde{X} = \theta] \\ & (zz')(\overline{X_{3i}}\,\langle \lceil \mathfrak{r} \rceil, \tilde{X}, z, z'\rangle. \\ & (z\,(\tilde{Y}).\mathtt{A}(\tilde{X} \leftarrow \theta') + z'\,().\mathtt{A}(\tilde{X})))\end{aligned}$$

$$\begin{aligned}\mathfrak{r}_{B(\psi)}(\tilde{Y}) := {} & [\tilde{Y} = \psi] \\ & Y_{3j}\,(n, \tilde{Z}, z, z').[n = \lceil \mathfrak{r} \rceil][\tilde{Z}, \tilde{Y} = \theta, \psi] \\ & \bar{z}\,\langle \tilde{Y}\rangle.\mathtt{B}(\tilde{Y} \leftarrow \psi')\,;\,\overline{z'}\,\langle\rangle.\mathtt{B}(\tilde{Y})\end{aligned}$$

We observe that the binders $(\tilde{x})(\cdot)$ in the left hand side of $\mathfrak{r}$ are not encoded in π-calculus. By the third clause of Definition 7, these names may only be edge names in $\mathcal{E}$, and since all $m\kappa$-solutions are supposed to be graph-like, these names may only occur in the agents $A(\theta)$ and $B(\psi)$. Therefore, by definition of $[\![\cdot]\!]_\pi$, they only occur in $\mathtt{A}(\tilde{X})$ and $\mathtt{B}(\tilde{Y})$ and one does not need to test whether these names are private. (Which is fortunate since it seems very unlikely that this is possible.) Thus, the edges deletions performed by $\mathfrak{r}$ are mimicked in the processes by the erasings done by the updates $\tilde{X} \leftarrow \theta'$ and $\tilde{Y} \leftarrow \psi'$. What is not mimicked is the erasure of the binder itself $\tilde{x}$. The correctness of $[\![\cdot]\!]_\pi$ will be established up to this garbage collection.

Case 3: $\mathfrak{r} = (\tilde{x})(A(\theta), B(\psi)) \to A(\theta')$, this case is analog to case 2 except that the successful continuation $\mathtt{B}(\tilde{Y} \leftarrow \psi')$ in $\mathfrak{r}_{B(\psi)}(\tilde{Y})$ is replaced with $\mathbf{0}$.

Case 4: $\mathfrak{r} = A(\theta) \to (\tilde{x})(A(\theta'), B(\psi))$. Since there is only one agent on the left, one does not need a synchronization and a τ move is enough. Recall that, by the (synth) clause, the interface ψ is complete, so (the first clause of the definition of updates never

applies and therefore) none of the Y_j occur in $\mathtt{B}(\tilde{Y} \leftarrow \psi)$, and the process definition below is well-defined:

$$\mathfrak{r}_{A(\theta)}(\tilde{X}) := [\tilde{X} = \theta]\ \tau.(\tilde{x})(\mathtt{A}(\tilde{X} \leftarrow \theta') \mid \mathtt{B}(\tilde{Y} \leftarrow \psi)).$$

Case 5: $\mathfrak{r} = A(\theta) \to (\tilde{x})(A(\theta'))$, this case is similar to the preceding one:

$$\mathfrak{r}_{A(\theta)}(\tilde{X}) := [\tilde{X} = \theta]\ \tau.(\tilde{x})(\mathtt{A}(\tilde{X} \leftarrow \theta')).$$

Case 6: $\mathfrak{r} = (\tilde{x})(A(\theta')) \to A(\theta)$, again this case is similar to the preceding one:

$$\mathfrak{r}_{A(\theta)}(\tilde{X}) := [\tilde{X} = \theta]\ \tau.(\mathtt{A}(\tilde{X} \leftarrow \theta')).$$

Case 7: $\mathfrak{r} = (\tilde{x})(\mathtt{L}) \to \mathbf{0}$, this case is similar to case 6 if there is only one agent and case 3 if there are two of them; in both cases one just has to replace the successful continuation $\mathtt{A}(\tilde{X} \leftarrow \theta')$ with $\mathbf{0}$.

Case 8: $\mathfrak{r} = \mathbf{0} \to \mathtt{R}$, in this case we must define the process $\mathfrak{s}_{\mathtt{R}}$. There are two subcases: either $\mathtt{R} = (\tilde{x})(A(\theta))$ or $\mathtt{R} = (\tilde{x})(A(\theta), B(\psi))$, where θ and ψ are complete. Accordingly, $\mathfrak{s}_{\mathtt{R}}$ is defined as

$$\begin{aligned} \mathfrak{s}_{(\tilde{x})(A(\theta))} &:= (\tilde{x})(\mathtt{A}(\tilde{X} \leftarrow \theta)) \\ \mathfrak{s}_{(\tilde{x})(A(\theta),B(\psi))} &:= (\tilde{x})(\mathtt{A}(\tilde{X} \leftarrow \theta) \mid \mathtt{B}(\tilde{Y} \leftarrow \psi)). \end{aligned}$$

7.3. Examples

In this subsection, we review three examples illustrating various aspects of the translation explained above. Let us consider first the following "swap" interaction:

$$\mathfrak{s} = A(1^r), B(1^s) \to A(1^s), B(1^r)$$

with $r, s \in \mathcal{G}$, and $\mathfrak{s}(A) = \mathfrak{s}(B) = 1$. This interaction satisfies Definition 7 but it is not monotonic or antimonotonic because none of the sides is connected. We will modify it to be monotonic just below. As it is, the respective contributions of $\mathfrak{s}$ to the behavior of the processes A and B are:

$$\begin{aligned} \mathfrak{s}_{A(1^r)}(X_0, X_1, X_2) := {} & [X_1 = 0][X_2 = 1](z_0 z_1) \\ & \overline{\upsilon_{\mathfrak{s}}}\,\langle X_0, X_1, X_2, z_0, z_1\rangle. \\ & (z_1\,(Y_0, Y_1, Y_2).\mathtt{A}(Y_0, 0, 1) + z_0\,().\mathtt{A}(X_0, X_1, X_2)), \end{aligned}$$

$$\begin{aligned} \mathfrak{s}_{B(1^s)}(Y_0, Y_1, Y_2) := {} & [Y_1 = 0][Y_2 = 1] \\ & \upsilon_{\mathfrak{s}}\,(X_0, X_1, X_2, z_0, z_1).[\varnothing] \\ & (\overline{z_1}\,\langle Y_0, Y_1, Y_2\rangle.\mathtt{B}(X_0, 0, 1)\,;\, \overline{z_0}\,\langle\rangle.\mathtt{B}(Y_0, Y_1, Y_2)). \end{aligned}$$

No cross-check is needed in this case, hence the empty match $[\varnothing]$, but each process needs the name carried by the other.

As said, by introducing a slight variation on this interaction:

$$\mathfrak{s}' = A(1^r + 2), B(1^s + 2) \to (x)(A(1^s + 2^x), B(1^r + 2^x))$$

with $x \in \mathcal{E}$, $r, s \in \mathcal{G}$, and $\mathfrak{s}(A) = \mathfrak{s}(B) = 2$, we obtain a monotonic $m\kappa$-interaction. The corresponding terms in A and B now become

$$[X_1 = 0][X_2 = 1][X_3 = v][X_4 = 0][X_5 = 0](z_0 z_1)$$
$$\overline{v_{\mathfrak{s}'}}\,\langle X_0, z_0, z_1\rangle.$$
$$(z_1\,(x, Y_0).\mathtt{A}(Y_0, 0, 1, x, 0, 2) + z_0\,().\mathtt{A}(X_0, X_1, X_2, X_3, X_4, X_5)),$$

$$[Y_1 = 0][Y_2 = 1][Y_3 = v][Y_4 = 0][Y_5 = 0]$$
$$v_{\mathfrak{s}'}\,(X_0, z_0, z_1).[\varnothing]$$
$$((x)(\overline{z_1}\,\langle x, Y_0\rangle.\mathtt{B}(X_0, 0, 1, x, 0, 2))\,;\,\overline{z_0}\,\langle\rangle.\mathtt{B}(Y_0, Y_1, Y_2, Y_3, Y_4, Y_5)).$$

We have slightly shortened the translation by only sending *relevant* names: x, Y_0 from B to A and X_0 from A to B, and not the logs and type indications. This is something which can always be done.

As a last example, let us illustrate the cross-checking part of the encoding. Consider the following antimonotonic "unbind" reaction:

$$\mathfrak{r} = (x)(A(1^{x,3}), B(1^{x,3})) \to A(1), B(1),$$

where $x \in \mathcal{E}$, $\mathfrak{s}(A) = \mathfrak{s}(B) = 1$. Sending the only relevant name, here X_0, as we did in the preceding example, we find the corresponding contributions in A and B:

$$[X_1 = 3][X_2 = 2](z_0 z_1)$$
$$\overline{v_{\mathfrak{r}}}\,\langle X_0, z_0, z_1\rangle.(z_1\,().\mathtt{A}(v, 0, 0) + z_0\,().\mathtt{A}(X_0, X_1, X_2)),$$

$$[Y_1 = 3][Y_2 = 2]$$
$$v_{\mathfrak{r}}\,(X_0, z_0, z_1).[X_0 = Y_0](\overline{z_1}\,\langle\rangle.\mathtt{B}(v, 0, 0)\,;\,\overline{z_0}\,\langle\rangle.\mathtt{B}(Y_0, Y_1, Y_2))$$

and here, in contrast with the two preceding examples, one sees that B has to verify whether his x is the same as A's x, a task which is performed by the cross-check $[X_0 = Y_0]$.

7.4. Observations and Correctness

To conclude, we state the correctness properties of our encoding.

Given a set of $m\kappa$-interactions $\mathfrak{R}$, let us write $A(\theta) \downarrow_{\mathfrak{R}} \mathfrak{r}$ if $\mathfrak{r}$ is a *binary monotonic* interaction in $\mathfrak{R}$, and $A(\theta)$ matches one of the reactants of $\mathfrak{r}$. This observation relation extends to arbitrary $m\kappa$-solutions as follows:

$$\mathsf{S}, \mathsf{T} \downarrow_{\mathfrak{R}} \mathfrak{r} \qquad \text{if } \mathsf{S} \downarrow_{\mathfrak{R}} \mathfrak{r} \text{ or } \mathsf{T} \downarrow_{\mathfrak{R}} \mathfrak{r}$$
$$(x)(\mathsf{S}) \downarrow_{\mathfrak{R}} \mathfrak{r} \qquad \text{if } \mathsf{S} \downarrow_{\mathfrak{R}} \mathfrak{r}$$

Such observations are often called *barbs*.

Let us write $P \to Q$ to abbreviate $P \xrightarrow{\tau} Q$, and denote the transitive closure of $\to$ by $\to^{\star}$. We also define barbs on the π-calculus side:

$$P \downarrow x := \exists Q\, P \xrightarrow{\mu} Q$$

where μ is an *input or an output* action on a name x.

Finally, let $\equiv$ be the usual least congruence over π-calculus closed under renaming of bound variables (α-equivalence), making " $\mid$ " associative and commutative with $\mathbf{0}$ as neutral element, and satisfying the scope laws:

$$\begin{array}{lll}(x)(y)P & \equiv & (y)(x)P,\\ (x)P & \equiv & P \quad \text{if } x \notin \mathrm{fn}(P),\\ (x)P \mid Q & \equiv & (x)(P \mid Q) \quad \text{if } x \notin \mathrm{fn}(Q).\end{array}$$

Recall that $\upsilon_{\mathfrak{r}}$ is the name associated to a binary monotonic interaction $\mathfrak{r}$.

Theorem 6. *Let* $(\mathfrak{S}, \to)$ *be an m*κ*-system, and* S *be a closed m*κ*-solution*:
1. *if* $[\![\mathsf{S}]\!]_\pi \downarrow x$ *then* $x = \upsilon_{\mathfrak{r}}$ *for some binary monotonic interaction* $\mathfrak{r}$;
2. $\mathsf{S} \downarrow_{\mathfrak{R}} \mathfrak{r}$ *if and only if* $[\![\mathsf{S}]\!]_\pi \downarrow \upsilon_{\mathfrak{r}}$;
3. *if* $\mathsf{S} \to \mathsf{T}$ *then* $[\![\mathsf{S}]\!]_\pi \to^* \equiv [\![\mathsf{T}]\!]_\pi$;
4. *if* $[\![\mathsf{S}]\!]_\pi \to^* Q$, *then there exists* T *such that* $Q \to^* \equiv [\![\mathsf{T}]\!]_\pi$, *and* $\mathsf{S} \to^* \mathsf{T}$.

Proof (Sketch). One has to suppose that S is closed in order not to observe the edge names (see the X_{3i}, Y_{3j} in case 2). That said, points 1, 2 are obvious.

One also has to suppose S to be graph-like (which is the default assumption since the end of section 5) else one has problems with "news" on the left hand side (see the discussion at the end of case 2).

Points 3 and 4 are proved by establishing that for any Q such that $[\![\mathsf{S}]\!]_\pi \to^* Q$, there are three possible kinds of transitions: either a pair of processes (on a public channel $\upsilon_{\mathfrak{r}}$ or a private channel X_{3i}) going to an "unstable" state where one of them cross-checks both interfaces, or a success signal (on a private z channel) where an interaction is finalized, or a failure signal (on a private z' channel) where a pair of processes rolls back to a previous state. □

This theorem is more powerful than the corresponding result for the encoding of κ-calculus into $m\kappa$-calculus. Specifically, item 4 ensures that no deadlock is introduced by the encoding of the rules. Besides, item 2 meshes well with the intuition that one cannot observe an antimonotonic interaction which is an internal event. An easy consequence of all items together, is the weak barbed bisimilarity [20] of S and $[\![\mathsf{S}]\!]_\pi$.

8. Conclusion

We have presented a coarse-grained calculus of proteins and worked out a formalization of the lactose operon illustrating the ease and the precision with which our language can describe protein interactions and similar basic events of biological systems such as synthesis and even metabolite transformation. The process-algebraic notation which we choose, with its explicit edge residuals and built-in treatment of name generation seems elegant enough if one compares it with a graph-based formalism and in particular allows for a clean definition of the notion of monotonic reaction.

As a dynamic annotation language κ might be useful for the different purposes of archiving, playing and comparing models. But beyond its representational abilities,

κ also has an important structural property which we called self-assembly, and we have made this a theorem. Namely that there is a finer-grained calculus with only binary interactions and very limited additional local structure in which one can encode the coarser-grained κ-reactions. Here the notational investment really pays off and let the simplicity of the encoding be seen through the syntax.

Yet, and despite its spartan syntax, the finer-grained calculus might still seem to endow agents with too much intelligence to be biologically meaningful, and one might well wonder if our effort to explain the high-level reactions by incorporating them *in* the proteins has not led us contemplating unrealistically talented proteins. We do not think so. One way of understanding the combinatorial power of the agents, which is not enormous anyway, is to see it as a digital translation of the combinatorics that true proteins have, because they are embedded in space. Forward engineering of biological systems focuses on the analysis and construction of the basic components one can engineer in biological systems [14]. Our self-assembly result seems a valuable step in understanding *another* aspect of bio-computing that is how in a world where asynchrony is the norm, by using low-level binary synchronization events as building blocks, one can engineer arbitrary synchronizations.

References

[1] G. Berry, G. Boudol, The chemical abstract machine, Theoret. Comput. Sci. 96 (1992) 217–248.
[2] R. Bruni, C. Laneve, U. Montanari, Orchestrating transactions in join calculus, in: Proc. CONCUR'02. Lecture Notes in Computer Science, Vol. 2421, Springer, Berlin, 2002, pp. 321–337.
[3] L. Cardelli, Brane calculi, in: Proc. BIO-CONCUR'03, Marseille, France, Electronic Notes in Theoretical Computer Science, Elsevier, Amsterdam, 2003, to appear.
[4] L. Cardelli, P. Gardner, G. Ghelli, Manipulating trees with hidden labels, in: Proc. FOSSACS 2003, Lecture Notes in Computer Science, Vol. 2620, Springer, Berlin, 2003, p. 216–232.
[5] M. Chiaverini, V. Danos, A core modeling language for the working molecular biologist, in: Proc. CMSB'03, Lecture Notes in Computer Science, Vol. 2602, Springer, Berlin, 2003, p. 166.
[6] V. Danos, J. Krivine, Formal molecular biology done in CCS, in: Proc. BIO-CONCUR'03, Marseille, France, Electronic Notes in Theoretical Computer Science, Elsevier, 2003, to appear.
[7] V. Danos, G. Laneve, Core formal molecular biology, in: Proc. 12th European Symp. on Programming (ESOP'03), Warsaw, Poland, Lecture Notes in Computer Science, Vol. 2618, Springer, Berlin, 2003, pp. 302–318.
[8] V. Danos, C. Laneve, Graphs for formal molecular biology, in: Proc. First Internat. Workshop on Computational Methods in Systems Biology (CMSB'03), Rovereto, Italy, Lecture Notes in Computer Science, Vol. 2602, Springer Berlin, 2003, pp. 34–46.
[9] R. Diestel, Graph Theory, Springer, New York, 2000.
[10] F.L. Fessant, Jocaml: Conception et implémentation d'un langage à agents mobiles, Ph.D. Thesis, Ecole Polytechnique, France, 2001.
[11] W. Fontana, L.W. Buss, The barrier of objects: from dynamical systems to bounded organizations, in: J. Casti, A. Karlqvist (Eds.), Boundaries and Barriers, Addison-Wesley, Reading, MA, 1996, pp. 56–116.
[12] C. Fournet, G. Gonthier, The reflexive chemical abstract machine and the join-calculus, in: 23rd ACM Symp. on Principles of Programming Languages (POPL'96), 1996.
[13] D.T. Gillespie, Exact stochastic simulation of coupled chemical reactions, J. Phys. Chem. 81 (1977) 2340–2361.
[14] J. Hasty, D. McMillen, J.J. Collins, Engineered gene circuits, Nature 420 (2002) 224–230.
[15] T. Inada, K. Kimata, H. Aiba, Mechanism responsible for glucose-lactose diauxie in Escherichia Coli: challenge to the cAMP model, Genes Cells 1 (3) (1996) 293–301.

[16] J.W. Kimball, Kimball's biology pages, Online Biology Textbook, 2003.
[17] E. Klavins, Automatic synthesis of controllers for assembly and formation forming, in: Proc. Internat. Conf. Robotics and Automation, 2002.
[18] K.W. Kohn, Molecular interaction map of the mammalian cell cycle control and DNA repair systems, Mol. Biol. Cell (10) (1999) 2703–2734.
[19] Y. Lafont, Interaction combinators, Inform. and Comput. 137 (1) (1997) 69–101.
[20] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes I and II, Inform. and Comput. 100 (1992) 1–41, 42–78.
[21] J. Monod, F. Jacob, General conclusions: teleonomic mechanisms in cellular metabolism, growth and differentiation, Cold Spring Harbor Symp. on Quantitative Biology: Cellular Regulatory Mechanisms, Vol. 26, 1961, pp. 389–401.
[22] C. Priami, Stochastic pi-calculus with general distributions, in: CLUP (Ed.), Proc. PAPM 96, 1996.
[23] C. Priami, A. Regev, E. Shapiro, W. Silverman, Application of a stochastic name-passing calculus to representation and simulation of molecular processes, Inform. Process. Lett. (2001).
[24] A. Regev, E.M. Panina, W. Silverman, L. Cardelli, E. Shapiro, Bioambients: an abstraction for biological compartments, Theoret. Comput. Sci. (2003) to appear.
[25] A. Regev, E. Shapiro, Cells as computation, Nature 419 (2002).
[26] A. Regev, W. Silverman, E. Shapiro, Representation and simulation of biochemical processes using the π-calculus process algebra, in: R.B. Altman, A.K. Dunker, L. Hunter, T.E. Klein (Eds.), Pacific Symp. on Biocomputing, Vol. 6, World Scientific Press, Singapore, 2001, pp. 459–470.
[27] I.H. Segel, Enzyme Kinetics: Behavior and Analysis of Rapid Equilibrium and Steady-State Enzyme Systems, Wiley, New York, 1975.
[28] A. Unypoth, P. Sewell, Nomadic Pict: correct communication infrastructure for mobile computation, in: Proc. POPL 2001, 2001.