Computational self-assembly

Pierre-Louis Curien, Vincent Danos, Jean Krivine, Min Zhang

Computational self-assembly

Pierre-Louis Curien[*] Vincent Danos[†] Jean Krivine[‡] Min Zhang[§]

January 18, 2007

Abstract

The object of this paper is to appreciate the computational limits inherent in the combinatorics of an applied concurrent (aka agent-based) language κ.

That language is primarily meant as a visual and concise notation for biological signalling pathways. Descriptions in κ, when enriched with suitable kinetic information, generate simulations as continuous time Markov chains. However, κ can be studied independently of the intended application, in a purely computational fashion, and this is what we are doing here.

Specifically, we define a compilation of κ into a language where interactions can involve at most two agents at a time. That compilation is generic, the blow up in the number of rules is linear in the total rule set size, and the methodology used in deriving the compilation relies on an implicit causality analysis. The correctness proof is given in details, and correctness is spelt out in terms of the existence of a specific weak bisimulation. To compensate for the binary restriction, one allows components to create unique identifiers (aka names). An interesting by-product of the analysis is that when using acyclic rules, one sees that name creation is not needed, and κ can be fully reduced to binary form.

1 Introduction

Our knowledge of the combinatorial intricacies of signalling pathways is rapidly evolving. Kohn's molecular maps represent the earliest efforts of tying together mechanistic details of pathways into a format that enables sharing, discussion, revision, and supports quantitative modelling [10, 11]. Similar projects by Oda, Kitano, and collaborators provide an extensive account of the EGF and TOLL receptors pathways [14, 13]. Yet, none of these projects provides executable descriptions, nor do they express unambiguously or conveniently the combinatorial detail at hand.

The κ language was introduced a few years ago (and named κ as a reference to Kohn maps) in an attempt to partly formalise signalling, and give a better answer to that same set of questions [4]. That language goes in the community of modellers under the name of the BioNetGen language (BNG), as it was discovered independently, and developed into a modelling framework [9, 2, 6, 5]. Both languages are essentially the same and will be taken as synonyms in this paper. BNG

[*]CNRS & Univ. Paris 7
[†]CNRS & Univ. Paris 7 & Plectix Bio Systems
[‡]Plectix Bio Systems
[§]Univ. Shanghai

implements a natural extension of Gillespie's algorithm [8, 7] that given kinetic rates turns the non-deterministic transition system associated to a BNG set of rules into a continuous time Markov chain (or a differential system if the set of reachable complexes is finite and can be constructed), and that is obviously of interest for numerical simulation.

That language being intuitive, graphical, and quantitative, it seems a good start in the development of an environment for building complex models of signalling. Because it is rule-based, ie based on a contextual semantics, it allows for concise definitions of models, which, if one were to use a flat enumeration of chemical reactions, would be difficult to write down by hand, and even more difficult to modify. Even a simple EGF model[1] generates hundreds of different complex components, and thousands of flat reactions. Another valuable consequence of dealing with structured agents is that one can define the attendant notion of causality (when a rule appears necessarily before another one in a given trace) as in the traditional event structure representation of processes [20]. This leads to rather informative representations of traces as partial-time minimal configurations, giving the user a better handle on the overwhelming number of traces.

Coming closer to the subject of the present paper, there was the further ambition in defining κ, to set a framework in which some computational aspects of biological information processing could be studied. Leaving aside the representation aspects of κ, we see it here as a purely computational graph-rewriting model of a certain type, and try to understand whether the ability of rewriting any number of agents can be dispensed with, and one can restrict oneself to (at most) binary interactions and still obtain the same range of behaviours. This is what we call the *self-assembly* question, although we use the term merely as an analogy since our result does not deal with the physics and engineering issues, and self assembly has to be understood here as a distributed programming question using an idealised medium, as the title is meant to suggest.

We solve the question to the positive although, and importantly, one cannot simply binarily restrict κ, and to recover full expressiveness, one has to endow the system with an additional computational mechanism, namely the ability to create unique identifiers. Under that condition, we are able to give a complete binary reduction of κ, and furthermore we offer an informal argument showing that it would probably be very difficult to do without name creation. This kind of negative result is notoriously tricky to set up, and we keep it informal and inconclusive, though we find it persuasive. Finally, one sees easily from the generic compilation, that tree-based rules (where rules never use cyclic complexes), actually don't need that additional name creation facility.

Our contribution can be set in different perspectives.

From the distributed algorithmics point of view this is a rather neat and efficient distributed implementation of a graph-rewriting grammar, with the natural granularity choice of having agents be the graph nodes. This breakdown of the grammar between agents (none of the agents actually know the global grammar rules) essentially amounts to a distributed consensus into binary asynchronous interactions. The trick is to use reversible rules so as to escape deadlocks, and that means keeping careful track of causality. Furthermore the proof is done in details, which is important for such rather large and non intuitive agent systems, and follows broadly the recent 'history category construction' [3].

From a process or agent-based language perspective,[2] this is a way of measuring the computational

[1]See subsection 2.1 for a brief presentation of that model taken from Ref. [1]

[2]By agent-based language, one usually means a language defining processes offering interactions with other processes,

power of κ, and one can easily derive a compilation down to π-calculus [12] from there (as shown in Ref. [4]). This relates to previous work using directly π-calculus to model pathways [19, 17, 18, 16], in that it gives a structured and systematic way to obtain such lower-level descriptions (as a compilation should) from a visual higher-level notation, should one need it.

But more importantly, that also says that there is a natural and naturally distributed task for which π-calculus is complete. Note that one does not mean to say the π-calculus is Turing complete (which it also is, and so is κ too as we will see soon), but complete in a sense which is much more difficult to make precise. This is we believe the first instance of such a result, however for reasons we discuss further in the conclusion, this is a difficult line of argument to tread, and there is for now no satisfying theorem to prove here. The conclusion also discusses in which sense this study may be relevant from a biological point of view.

2 κ

We start with a brief description of κ.

Suppose given a countable set of agent names $\mathcal{A}$, a countable set of sites $\mathcal{S}$, and a finite set of values $\mathbb{V}$. An *agent* is a tuple consisting in an agent name $A \in \mathcal{A}$, a finite set of sites $s \subseteq \mathcal{S}$, and a partial valuation in $\mathbb{V}^s$, called the agent *internal state*, and associating values to some of the agent sites. We write $\lambda(a)$ for the name of an agent a, $\sigma(a)$ for the set of its sites, and $\mu(a)$ for its internal state.

A *κ-solution* S is a set of agents, together with a partial matching on the set $\sum_{a \in S} \sigma(a)$ of all agent sites. In plain terms, agents can connect via their sites, but no site can be connected twice. A site which is (not) in the domain of the matching is said to be bound (free). One writes $(a, i), (b, j) \in S$ to express the fact that sites i, j in agents a, b are matched in S.

Every solution has an underlying graph structure, and we shall freely use graph-theoretic notions such as subgraph, connected component, path, etc., and rely mostly on graphical presentations of all the concepts involved. Following biological terminology, connected solutions are also called *complexes*.

An injection ϕ between solutions S and S' is an *embedding* if for all $a, b \in S$, $i, j \in \mathcal{S}$:

$\lambda(a) = \lambda'(\phi(a))$	names are preserved
$\sigma(a) \subseteq \sigma'(\phi(a))$	sites are preserved
$\mu(a)(i) = v \Rightarrow \mu'(\phi(a))(i) = v$	internal states are preserved
$(a, i), (b, j) \in S \Leftrightarrow (\phi(a), i), (\phi(b), j) \in S'$	edges are preserved

conditioned on the participants states, and resulting in redefining states and subsequent interaction offers for each participants. The implied dynamics may or may not be synchronous, and may or may not have a quantitative aspect such as defining a stochastic process or a differential system. Interactions may include two agents or more, agents may split in concurrent continuations and may dynamically create new means of interaction (one refers to this as mobility sometimes although no actual movement in space is implied). Note that processes are defined in terms of continuations subsequent to an interaction, so that their identity and lineage can be tracked through the evolution of a system. Agent-based models of biological systems are therefore naturally testable with respect to queries pertaining to the lineage of a biological entity, and causal dependencies in order to reach a given observable of interest.

A *signature* map is an assignment of a finite set of sites to each agent name. A solution S is said to be *complete* with respect to such a map $\sigma : \mathcal{A} \rightarrow \wp(\mathcal{S})$, if for all $A \in \mathcal{A}$, $a \in S$, if $\lambda(a) = A$, then $\sigma(a) = \sigma(A)$.

We suppose such a signature map fixed once and for all.

Atomic actions one wishes to perform on a solution $\mathcal{S}$ are: changing the value of some site, creating/deleting an edge between two sites, and creating/deleting an agent. Deletion of an agent entails deleting all edges the node may share.

An *action* over S is a sequence of such atomic actions over S. A *rule* is a pair (S, α) of a solution S (the rule pattern, or left hand side) and an action α over S (the rule action). The result of applying the rule action to the rule guard is written $\alpha \cdot S$ (the rule result, or right hand side).

A rule (S, α) is said to be *atomic* if α is atomic; *connected* if either S, or $\alpha \cdot S$ is; *binary* if S has at most two agents; *monotonic* if α uses no edge or agent deletion, and *anti-monotonic* if α uses no edge or agent creation.

Given an embedding $\phi : S \rightarrow S'$, an action α on S transfers over to S' in the obvious way, and one writes $\phi(\alpha) \cdot S'$ for the result of the action α on S' via ϕ. Since S may have automorphisms, specifying only the domain of ϕ does not suffice to determine in general the result.

Definition 1 *A set R of rules defines a labelled transition relation over complete solutions:*

$$S' \longrightarrow_{\phi}^{S,\alpha} \phi(\alpha) \cdot S' \tag{1}$$

with $(S, \alpha) \in R$, and ϕ an embedding from S into S'.

A rule instance, or a rule application, therefore consists in identifying an embedding ϕ of the rule pattern into a solution, and applying the rule action with its domain being renamed by ϕ.

2.1 A biological example

Fig. 2 displays a sample set of biologically plausible and atomic rules, where rule actions are represented as dotted lines. Specifically, directed edges represent modifications (phosphorylation induced by the receptor kinase domain represented as a red circle, and inducing a state change represented as a solid black circle); undirected ones represent bindings. Those rules are drawn from a simplified model of the early EGFR (epidermal growth factor receptor) pathway [1].

Fig. 1 shows an example of a complex which can be constructed using the said rules, eg starting with a set of unphosphorylated disconnected agents (a reasonable initial state for a signalling cascade). The actual system is known to have about ten variant signals (EGF) and receptors (EGFR) [14], so the model is really a simplification of the pathway.

We use sometimes an in-text *process notation* for solution and rules, eg the first two rules in the left column of Fig. 2 are written:

$$\begin{array}{l}
\text{EGFR}(l, r) + \text{EGF}(r) \rightarrow \text{EGFR}(l^x, r) + \text{EGF}(r^x) \\
\text{EGFR}(l^{x_1}, r^y, Y1148^u) + \text{EGF}(r^{x_1}) + \text{EGFR}(l^{x_2}, r^y) + \text{EGF}(r^{x_2}) \rightarrow \\
\text{EGFR}(l^{x_1}, r^y, Y1148^p) + \text{EGF}(r^{x_1}) + \text{EGFR}(l^{x_2}, r^y) + \text{EGF}(r^{x_2})
\end{array}$$

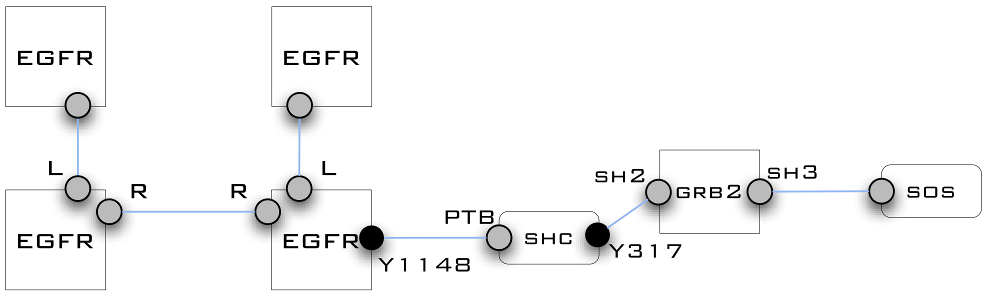


Figure 1: Nodes correspond to proteins, sites correspond to protein domains or/and amino-acid residues susceptible of post-translational modifications (solid black).

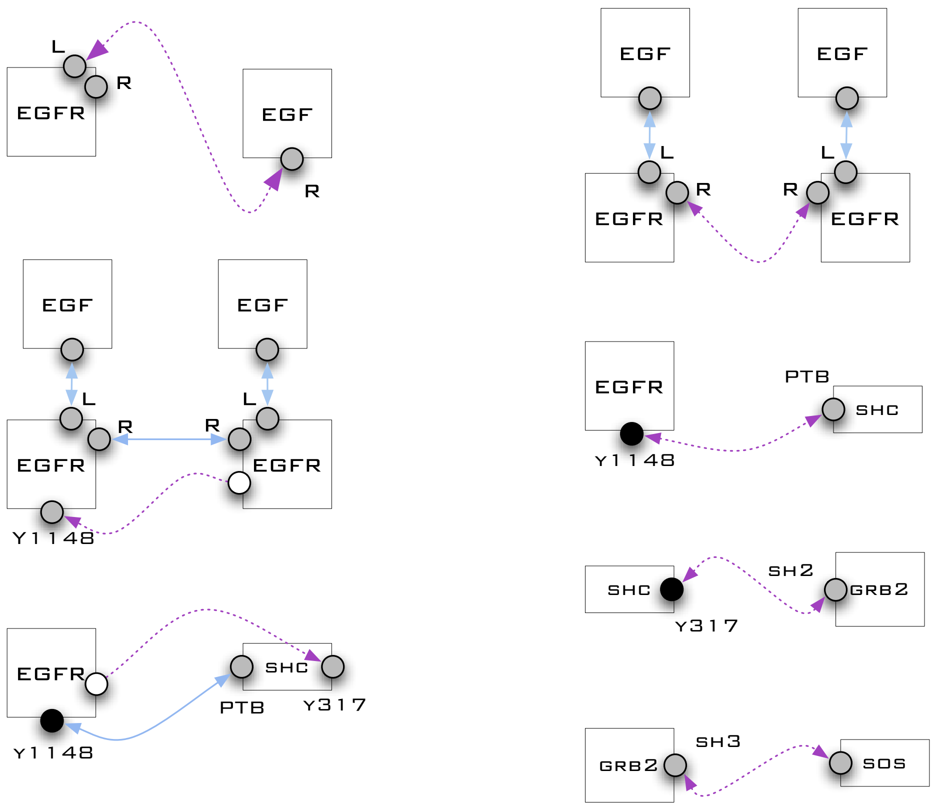


Figure 2: The left column includes signal/receptor binding, receptor phosphorylation, phosphorylation of the adaptor Shc; the right column shows receptor dimerisation, binding of the adapter Shc, binding of Grb2, and binding of Sos. The EGFR kinase site responsible for the various phosphorylations is represented as a solid white circle.

In that notation bindings and internal states are indicated as superscripts to their sites, the rule pattern is the left hand side, and the rule action is to be deduced as the difference between the right and left hand sides. For instance in the first rule, the action consists in creating an edge, which is represented here by the name x being shared as a superscript by sites l in EGFR, and r in EGF.

Both notations carry almost the same information. Actually, one needs to add to the process notation a mapping from the left hand side agents to the right hand side ones, so the graphical notation is both more intuitive and more precise. Furthermore the graphical notation makes it easy to add some biologically meaningful information such as the EGFR kinase site (solid white in Fig. 2) being responsible for the flipping of the internal state associated to EGFR(Y1148), and Shc(Y317) (see the graphical version of the second, and third rule in the first column above). That information is lost in the process notation, however it is not needed for the semantics. Although the process notation is heavier it is sometimes convenient, and closer to the notation used in the implementation.

3 Self-assembly

From a process calculus point of view, where each agent is thought of as an independent computation, κ allows for arbitrary many agents to synchronise in a rule (eg four agents are needed in the EGFR dimerisation rule above). A natural question is whether one could obtain the same behaviours, using only binary or unary rules. This is not unlike asking in chemistry how one decomposes a complex reaction in elementary ones, which is something one usually does for the sake of assigning kinetics to that complex reaction.

3.1 Turing machines

Before we embark on this reduction to binary interactions, and as an exercise, let us show how one can encode Turing machines using only binary or unary rules. This also stresses the fact that the expressivity result we are after has little to do with the traditional way of assessing sequential computational complexity. It is all about restricting the communication means.

So suppose given an alphabet Σ, a state space Q, and a transition function δ. To represent a tape unit element, we use one agent type, which we keep nameless, with 'left' and 'right' binding sites l, and r, and 'up' and 'down' sites u, d used to hold a value in $\Sigma \times (Q + \{*\})$ and kept free at all time. See Fig. 3. Both the alphabet Σ and the state space Q are finite sets so it is possible to encode them as internal states. The tape is represented as a finite chain of agents. We write B for the blank symbol, use $*$ as an idleness flag, and the q value both as a representation of the current state, and an activity flag. Only one tape element is active at a time (the one the 'head' currently points at), or to be more precise, the set of rules below preserves this invariant.

A left transition (see also Fig. 3) and a right transition, $\delta(a, q) = (b, q', \rightarrow)$, and $\delta(a, q) = (b, q', \leftarrow)$

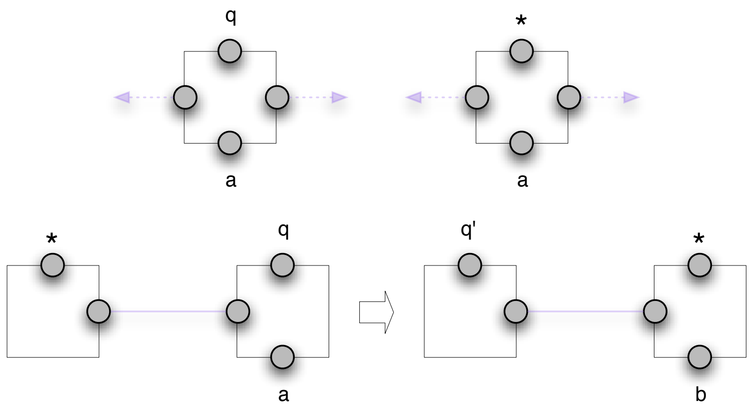


Figure 3: Tape unit elements (top row): the upper site is used store the machine state, when the tape unit is active (as in the left agent), when it is not ones uses $*$ (as in the right agent); the lower site is there to hold a tape symbol; the left and right sites are used to chain together tape unit elements. Left transition rule (bottom row): only internal states are modified, and the rule is not atomic since it modifies at least two internal states.

translate respectively in process notation as:

$$\begin{array}{l}
\langle *, r^x\rangle + \langle l^x, d^a, u^q\rangle \to \langle u^{q'}, r^x\rangle + \langle l^x, d^b, *\rangle \\
\langle d^a, q, r^x\rangle + \langle l^x, *\rangle \to \langle d^b, *, r^x\rangle + \langle l^x, u^{q'}\rangle \\
\langle r\rangle + \langle l, u^q\rangle \to \langle r^x\rangle + \langle l^x, u^q\rangle \\
\langle u^q, r\rangle + \langle l\rangle \to \langle u^q, r^x\rangle + \langle l^x\rangle \\
\to \langle l, u^*, d^B, r\rangle
\end{array}$$

The three other rules are there to handle unbounded computations, providing means to extend the tape on both sides, and to produce more blank tape unit elements. Those last three rules could be combined with the upper two to cope with the case where there is no left (right) neighbour to move the head to.

In the latter case the encoding halts effectively when the machine does, else the tape will still grow on both ends although the head (ie the activity token) will no longer move. Were there more than one active agent on the tape, the system would behave as an asynchronous multiheaded TM, with heads being on the same tape.[3] Since the encoding is really simple, one can think of binary κ as embedding Turing machines, and that is sometimes a useful thing to keep in mind.

3.2 $m\kappa$

Since one can encode Turing machines in binary κ, it seems encoding κ itself should be easy. However, as said above, one should not confuse sequential expressiveness in the sense of Turing ma-

[3]Mind that if one was not testing for the activity token in the two tape extension rules, a cyclic tape could form by tying both ends. Note also that rules are using only radius 1 agent views here, but crucially the first two rules are not atomic, else information could not be passed around so easily.

chines, and the more elusive notion of communication expressiveness we have to deal with here. Actually, we have not been able to produce such an encoding without enriching first the agents range of behaviour. There may be very good reasons for this, and we will return to discussing this in the last section.

For now, the agent enrichment is the following. Firstly, one adds in a countable set $\mathcal{G}$ of *group names*, and allows agents to include in their internal states one of these group names. Graphically this is expressed as an additional label to the agent. Secondly, a rule action can now incorporate the creation of a new group name. This is the key and only additional computational feature needed.[4]

We call that new language $m\kappa$.

3.3 Scenarios

The compilation below includes as a first step a translation of κ solutions into $m\kappa$ ones. That translation preserves the natural granularity, introduces no auxiliary agents, and in fact represents a solution as itself, just enlarging the agent internal state space. The second and trickier step is to break down a rule into a set of binary rules.

Here is an account of the method we use, which we then proceed to describe in details, and to prove correct in the next section. Suppose first that the rule under consideration is connected and monotonic. The idea is to replace the instantaneous recognition of the rule pattern, as provided by the embedding ϕ in (1), by a gradual and distributed construction of such an embedding. Group names, as the name suggests, are used at that point to build a transient cooperative structure corresponding to a partial embedding, partially acted on by the rule, as new nodes and edges are being recruited. Specifically, the group name is created by some agent initiating the exploration, and at any point during exploration, agents of a same group embed a (connected) subgraph of the rule right hand side (this statement is formalised later as lemma 4). If and when that embedding is eventually completed, agents shed their group names, ie leave the group, and the solution is there again an ordinary κ solution, and the net effect is that the rule was applied. To cater for the case where that gradual exploration fails to recognise the intended rule pattern, all rules the success of which is not guaranteed are made reversible.

To organise the construction of that partial embedding one uses a scenario, which is a statically defined structure that is not defined in a unique way, depends on the rule of interest, and will define how to pass information around, and in particular who is to initiate the exploration.

Definition 2 (Scenario) *Let (S, α) be a monotonic connected rule, a* scenario *for S, α is a triple $(\mathcal{F}, \mathcal{T}, in)$ such that:*
- $\mathcal{F}$ is an acyclic orientation of $\alpha \cdot S$
- $\mathcal{T}$ is a tree spanning $\mathcal{F}$ which is a sub directed graph of $\mathcal{F}$
- in is the common root of $\mathcal{F}$ and $\mathcal{T}$
- and in is in S

[4]For the readers familiar with π-calculus, no computation can be done on names, further than comparing them for equality. So this additional feature is easily represented in π.

Such scenarios always exist, since $\alpha \cdot S$ is connected, and any such 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 obtained tree ordering. The last stipulation, namely that in be in S, prevents a scenario from choosing an agent which is to be created by the rule (so only exists in $\alpha \cdot S$).

The acyclic $\mathcal{F}$ places a constraint on how the partial embedding is gradually extended: it will start at in, and then proceed following the ordering implied by $\mathcal{F}$. The spanning tree $\mathcal{T}$ serves as a way of constraining further this ordering, and imposes a 'parental priority' whereby a node of $\mathcal{F}$ can only be included in the embedding under construction by its unique parent in $\mathcal{T}$.

One could generalise scenarios to use more than one initiator. One could also perhaps dispense with the spanning tree. We discuss this later.

The directed graph $\mathcal{F}$ can be presented as a map over sites, defined as:

$$\begin{array}{ll} \mathcal{F}(a,i) = (b,j) & \text{if there is an edge from } (a,i) \text{ to } (b,j) \text{ in } \mathcal{F} \\ \mathcal{F}(a,i) = \perp & \text{if } (a,i) \text{ is free in } \mathcal{F} \end{array}$$

We write $\mathcal{F}^*$ for the inverse of $\mathcal{F}$, and use the same notations for $\mathcal{T}$.

Definition 3 (Inputs and outputs) *A site is an* output *if it belongs to the domain of $\mathcal{F}$, and a (principal)* input *if it belongs to the range of ($\mathcal{T}$) $\mathcal{F}$. In other words, a site (a,i) is called an output if $\mathcal{F}(a,i) \neq \perp$, an input if $\mathcal{F}^*(a,i) \neq \perp$, and a principal input if $\mathcal{T}^*(a,i) \neq \perp$.*

Clearly this obtains for any node in $\mathcal{F}$ a partition of its sites into the principal input, the secondary inputs, and the outputs. Any of these three classes may be empty, but there must at leat one site for each agent (because $\mathcal{F}$ is connected).

We will use the following ordering on sites in the proof of correctness.

Definition 4 (signal ordering) *Define a binary relation over $\mathcal{F}$ sites, written $\prec$, as the smallest transitive relation such that:*

$$\begin{array}{lcl} \mathcal{F}(a,i) = (b,j) & \Rightarrow & (a,i) \prec (b,j) \\ (a,i) \textit{ input}, (a,j) \textit{ output} & \Rightarrow & (a,i) \prec (a,j) \end{array}$$

3.4 The rules

We suppose now given a monotonic connected rule and a scenario $(S, \alpha, \mathcal{F})$, and give a graphical description of the associated set of $m\kappa$ rules. To make notations less daunting, we suppose further that neither S nor $\alpha \cdot S$ have loops (edges from a node to itself), that α contains no state modification, and that $\mathcal{F}$ has no free sites. The techniques described here adapt easily to those cases, since loops, free site testing, and state modifications are purely local to an agent.

As said, the agents have no notion of their context, and have to explore it and see whether the κ rule applies. There is a first phase of *recruitment*, and a subsequent phase of *completion*. Recruitment begins with the *initiator* being activated via rule $init$; then contact rules fc and lc are used to extend the initial embedding, and the response rules $resp$ to report success back to the initiator. At the end of this first phase, the initiator knows there is a suitable embedding for S, and α has taken place.

So it shifts to the completion phase using rule ps, and the news is shipped to the other agents using rule pp until every agent finally exits using rule $exit$.

The recruitment phase may never come to a successful end, either because the embedding sought for does not exist (if it does, then it does uniquely per connected component of S), or because its image is partly recruited into other groups (competition with other rule instances). Therefore all rules within that phase are made reversible, and the system never gets trapped in partial inconclusive explorations (deadlocks).

The formal compilation is described in the following paragraphs:
- the unique initial rule: $init$ (§3.4.1)
- the recruitment rules where signal flows down $\mathcal{T}$: fc (§3.4.2)
- the secondary recruitment rules where signal flows down $\mathcal{F} \setminus T$: lc (§3.4.3)
- the response rules where signal flows back up $\mathcal{F}$: $resp$ (§3.4.4)
- the unique phase-shift rule: ps (§3.4.1)
- the post phase shift propagation rules: pp (§3.4.4), and $exit$ (§3.4.1)

The $m\kappa$ rules are represented in the left/right hand side manner, which is better suited to handle several modifications at each step; reversible rules use double sided arrows.

There is a number of conventions used in the graphics. The group name is written g, other names a, b, stand for agents of $\mathcal{F}$, and are referred to as *roles*. For any agent with role a, those sites that are represented in its rules are those being bound in $\mathcal{F}$, and the internal numerical states they bear are referred to as *logs*. To save space, agent names are not represented (they can be uniquely retrieved from their role in $\mathcal{F}$), and the site names are only defined in the explanation of the rule. Directed edges are as in $\mathcal{F}$.

Both the fc and lc (described right below) rules distinguish in the bottom agent input sites the principal one and the secondary ones, represented respectively as the agent top site, and the agent side sites. In the remaining rules, there is no need for such a distinction, and top sites represents any input, principal or not. Bottom sites always stand for the agent outputs.

Finally, and importantly, agent sites coloured grey are quantified *universally*, that is to say the relevant internal state condition or modification bears on all sites of the same sort, secondary inputs, or all inputs, or all outputs. For instance in the right hand side of rule $init$, all outputs of in are set to 0.

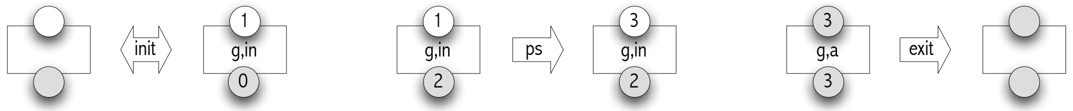
3.4.1 Initiation, phase shift, and exit

The first two rules $init$, and ps are specific to the initiator in in $\mathcal{F}$.

The left one is the initiation rule and is reversible. Since the agent is the root in $\mathcal{F}$, it has no secondary inputs. It does not have a principal one either, but we introduce here a *fictitious* site to make the formulation of invariants used in the correctness proof easier (see below). It is set to 1, while all outputs are set to 0. The group name g is created by the rule and assumed to be unique.

The middle rule is the phase shift. The root has collected all success signals (all the outputs are at 2) and shifts to the next phase; success is guaranteed here so the rule is not reversible.

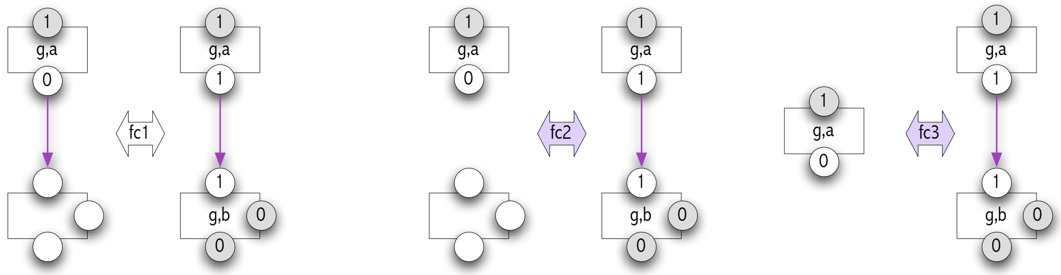
The right rule is the exit rule whereby when it has received and sent all termination messages (all the inputs and outputs are at 3), an agent leaves the group and all logs are erased. That rule is available to all agents with a role a in $\mathcal{F}$.



3.4.2 First contacts

A 'first contact' rule applies when $\mathcal{T}(a, i) = (b, j)$, where i is the top site, and j the bottom one. Those come in three different types: in *fc1* nothing is created, only logs are changed, whereas in *fc2* one has in addition to create the edge as stipulated by α, and in *fc3* one also has to create the b node.

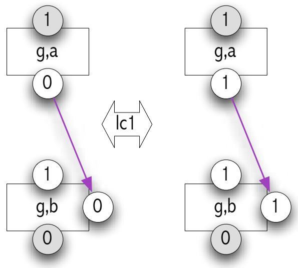
In all three cases, all of the top agent inputs have to be at 1, and the (a, i) output log is set to 1, the newly recruited bottom agent has its principal input log set to 1, while its other logs (secondary inputs and outputs) are all set to 0. The bottom agent is also labelled by its role b in $\mathcal{F}$, and the current group name g.

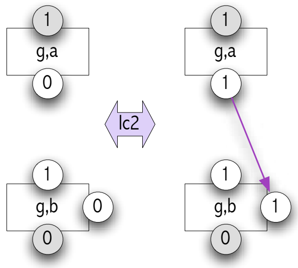


3.4.3 Later contacts

A 'later contact' rule applies when $\mathcal{F}(a, i) = (b, j)$, while $\mathcal{T}(a, i) = \bot$. So, an *lc* rule differs from an *fc* rule since it involves an edge to a secondary input of the bottom agent. As in the *fc* case all of the agent inputs must have log 1. Depending on whether that edge already exists in S, or not, one uses *lc1*, or *lc2*. In both cases the (a, i) output log and the (b, j) input log are set to 1. There is no third rule since by definition the node to connect to would already have been created at that stage by an *fc3* rule. Still by definition the bottom agent has already been recruited, and importantly the rule checks that both agents belong to the same group.

Such rules are not needed if $\alpha \cdot S$ has no cycles.



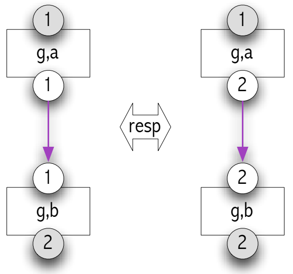


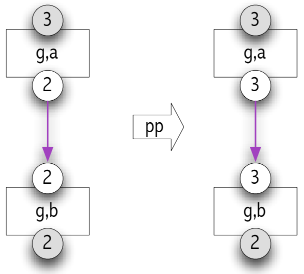
3.4.4 Responses, and propagation

The left rule is the response rule and is used to propagate upwards the local success signal (log 2) to the root. It makes no difference whether this signal is shipped back via a principal or secondary input of the bottom agent, so here there is no distinction between the principal input and the other inputs, and the bottom agent top site is any input. For the rule to apply, all the bottom agent outputs have to be at 2, and all the top agents input have to be at 1.

The right rule is dual to the left one, and propagates downwards the global success signal (log 3). For the rule to apply, all the bottom agent outputs have to be at 2, and all the top agent inputs at 3. Since one cannnot fail at that point, it is not reversible.

Note that both rules apply in particular when the bottom agent is a leaf in $\mathcal{F}$ and has no outputs.





3.5 Discussion

All the forward and backward rules above involve at most two agents as should be. Even though rules are modifying at most one edge at a time, they modify several logs, and except in the case of an especially simple $\mathcal{F}$ will not be atomic.

Except for $fc2$, $fc3$, and $lc2$, none of the rules have any impact on the underlying graph, and are merely passing information around by affecting only the agents internal states (the logs). Apart from name creation in $init$, only modification actions are used for this, so overall the compilation of a rule does not introduce any new edge/node creation or deletion (we will see that one may add some new edges when the rule is not connected). It is also interesting to notice that the agents don't 'know' which global κ rule is being attempted, all they know is how the embedding in S looks in their immediate (radius 1) neighbourhood, and the local modifications implied by α. A final thing to observe is that none of the rules but $init$ and $fc2$ have any inherent non-determinism, in the sense that once the top agent is chosen, the rule applies in a unique way (if it does at all).

The special fictitious input introduced for the initiator eases the correctness argument given in the next section. Apart from this, considering a special input for the root (and symmetrically a special output at leaves) is quite natural in view of a modular development of our techniques. Here we introduce only this site at the root, which is enough for our present purpose.

Finally, we also notice that the notion of group name can be replaced everywhere by a simple busy/free Boolean flag, except for the lc rules which test whether the two participants belong to the same group. One can therefore dispense with the group names if the rule under consideration is such that $\alpha \cdot S$ has no (undirected) cycle.

4 Correctness of self-assemby

We now discuss the mathematical properties of self-assembly, which culminate in the desired proof of correctness.

Let us write $[S]$ for the $m\kappa$ translation of a solution S, which is obtained by adding to each agent in S an empty internal state to hold the group name, and role during exploration, and a fictitious site to be used in the $init$ rule.

We also write $pre\text{-}ps$ for the set of forward rules $init$, fc, lc, and $resp$; $pre\text{-}ps^*$ for the set of backward rules $init^*$, fc^*, lc^*, and $resp^*$; and $post\text{-}ps$ for the set of rules pp, and $exit$.

The transition relations $\rightarrow_c$ and $\rightarrow_{m\kappa}$ are defined respectively as the union of $pre\text{-}ps^*$ and $post\text{-}ps$, and as the union of $pre\text{-}ps$, ps, and $post\text{-}ps$ (that is to say the union of all forward rules). The transition relation generated by all rules is simply written $\rightarrow$. For all of those relations, eg the last one, we denote its transitive closure as $\rightarrow^*$ (not to be confused with backward rules, eg fc^*).

A weak form of correctness was obtained earlier, namely that $S \rightarrow_\kappa S_1$ implies $[S] \rightarrow^* [S_1]$, using a subset of the rules above. This is easy to prove, one just applies the set of $m\kappa$-rules in algorithmic order: initiate, recruit, signal success to the root, signal success to all participating agents [4].[5]

It was also proved beforehand that, in a suitable sense, the compilation could not produce any wrong rewriting. However the one thing which was not proved, and which was actually clearly not true for the simplified rules, was that $m\kappa$ computations could not deadlock. With the present full compilation including the $pre\text{-}ps^*$ rules, we can now seek a stronger correctness property, namely, that any generated $m\kappa$ computation can at any moment be completed, either by undoing some steps of computation, or by bringing some others to their conclusion. This implies in particular the absence of deadlocks, and is formalized below (corollary 2).

[5]Group names, or reversible rules are not needed for simulation since one has free choice of the scheduling. Perhaps a useful way to think about the algorithm is to think about an NP problem; the simulation is similar to the verification of a solution, while the exploration part is internalising the search for a candidate solution. Correctness then consists in proving that a solution is always found if there is one, and no false solution is being discovered.

To make the reading easier we will denote κ solutions using symbols derived from S, while $m\kappa$ solutions will be written using symbols derived from T.

4.1 The relation $\rightarrow_c$ is confluent

We start by establishing the confluence of $\rightarrow_c$.

Lemma 1 (Strong Normalization) *The relation $\rightarrow_c$ is strongly normalizing.*

Proof: Let T be a solution, with n_i occurrences of log i $(i = 0, 1, 2, 3)$,and set $\rho(T) = p_0 n_0 + p_1 n_1 + p_2 n_2 + p_3 n_3$, for some natural numbers p_0, p_1, p_2, p_3 such that $0 < p_0 < p_1 < p_2 > p_3 > 0$. It is easily checked that if $T \rightarrow_c T'$ then $\rho(T') < \rho(T)$, and strong normalization follows. □

Lemma 2 (Local Confluence) *The relation $\rightarrow_c$ is locally confluent.*

Proof: Suppose $T \rightarrow_c T_1$, and $T \rightarrow_c T_2$. If the rules instances are such that their respective embedding codomains in T, say T_1, and T_2 are disjoint, then they clearly commute, because their associated actions have disjoint supports. The only non obvious case is *fc3** since this involves erasing an agent, which could potentially affect all neighbours by breaking edges to it, but the rule requires the bottom agent only connection to be that with the top agent, so no such interference can happen.

We have now to enumerate overlaps. One thing to notice is that agents of T_1 and T_2 are all holding one and the same group name: this is because all rules in $\rightarrow_c$ have their lhs agents labelled by a group name, which is the same in case the rule is binary, and since the two sets of agents intersect, the conclusion follows.

Let us prove first that none of the unary rules can overlap with any other one.
- Suppose T_1 matches the lhs of rule *init**, then the agent is the root of $\mathcal{F}$, so could only match a top agent in *fc**, *lc**, *resp**, which is impossible because of its output logs being at 0, or a top agent in pp, which is impossible because of the (fictitious) input log being at 1, or the unique agent of $exit$, which is impossible for the same reason. Therefore *init** does not overlap.
- Suppose now T_1 matches the lhs of rule $exit$, then none of the other rules in $\rightarrow_c$ has an input log at 3, except pp but then the outputs are not all at 3. So $exit$ cannot overlap either.

The remaining rules in $\rightarrow_c$ are binary, and deal each with one specific edge of $\mathcal{F}$. The ways in which those rules can overlap is restricted. Specifically, an agent that matches a bottom agent in a binary rule cannot at the same time match a top agent of any other rule.
- The bottom agent in rule pp has one input log at 2 which prevents him to embed as the top agent of another instance of pp (because one needs all inputs at 3), or as the top agent of all other binary rules in $\rightarrow_c$ (because those require all inputs at 1).
- The same argument can be made for the other binary rules *fc**, *lc**, *resp**.

Hence the only form of overlap is between binary rules, and both binary rules either share the same top agent *(i)*, or the same bottom one *(ii)*, or both *(iii)*.

In the special double overlap case *(iii)*, unless they are identical, the two rules must apply to two distinct edges. Indeed:
- an *fc** edge cannot be another *fc** edge since those edges are partitioned according to α, nor can

it be an lc^* one because an edge either belongs to $\mathcal{T}$ or not, nor can it be a $resp^*$ edge since the edge logs are either 1 or 2, or a pp one for the same reason;
- likewise an lc^* edge cannot be another lc^* edge because of α, nor can it be a $resp^*$ or a pp edge because of the edge logs;
- a $resp^*$ edge cannot be another $resp^*$ edge, they would be the same rule instance, or be a pp edge because of the top inputs logs.

In addition, rule $fc3^*$ can overlap only in mode *(i)*, since its bottom agent has only one bound site.

Let us consider the 'top' overlap case *(i)* first.
- Suppose one of the rules is pp. The top agent inputs are all at 3 for pp, and all at 1 in other rules (and there is always such an input site even if the top agent is the initiator), so pp can only top-overlap with itself, and that is a clearly confluent configuration.
- Suppose one of the rules is fc^*. Then the two bottom agents are distinct: fc^* cannot doubly overlap with another fc^* because there is by definition only one principal input per agent, nor with an lc^* because bottom secondary inputs are all at 0, nor with a $resp^*$ because all bottom inputs are at 0 or 1, and $resp^*$ demands that one of them is at 2. Now, in all such cases the two top agent embeddings intersect (and agree) exactly on the top agent input sites (which have to be set to 1), and their actions have disjoint support; therefore they commute.[6]
- Suppose one of the rules is lc^*. This time the two bottom agents can coincide resulting in a double overlap if the other rule is an lc^*, or a $resp^*$ (in the very specific case where the bottom agent is a leaf), in both cases confluence is immediate. Likewise if the two bottom agents are distinct, the same overlaps are possible and the rules commute.
- Finally suppose one of the rules is $resp$, then the only remaining case to consider is an overlap with itself, and it is readily seen to result in commuting rules.

We have now to consider the 'bottom' overlap case.
- Suppose then one of the rules is fc^*, then it cannot overlap with another fc^* since there is only one bottom principal input, nor can it overlap with any other rule since it fixes the bottom secondary inputs at 0, and all other kinds of rules ask for a bottom input site at 1 or 2.
- Suppose one of the two rules is pp. The other can be an lc^* (if the bottom agent is a leaf) or a $resp^*$, in which case the top agents are distinct, or another pp, and any such configurations commute.
- Suppose one of the two rules is an lc^*, then the other can be an lc^* too, or a $resp^*$, resulting in commuting configurations; all other cases were covered in the preceding subcases. □

Lemmas 1 and 2 obtain an interesting corollary.[7]

Corollary 1 (Confluence) *The relation $\rightarrow_c$ is confluent.*

We will write $c(T)$ for the unique $\rightarrow_c$ normal form of T.

[6]It is interesting by the way to notice that a backward first contact and a backward response in such a configuration can share a top agent, on the contact branch the recruitment signal is working backward, ie upwards, while on the response branch it is working backwards, ie downwards; this shows how asynchronous the propagation can be. Another noteworthy fact is that the two rules don't just happen to commute, they are concurrent according to the traditional residual-based definition of concurrency.

[7]This corollary implies in particular that the internal backtrack mechanism implemented by the set of backward rules $pre\text{-}ps^*$ is correct in the technical sense of Ref [3].

Local confluence being of the one-one type, as can be seen from the proof, it is enough to ensure (global) confluence, so strong normalisation is not really needed, and all $\rightarrow_c$ reductions to normal form have same length.[8]

4.2 The principal lemma

Say an $m\kappa$ solution T is *nice* if an agent in T has a group name iff it has a role iff it has no empty logs; and if any site i in T with a log in $\{1, 2\}$, is bound to a site j with the same log.

Lemma 3 *The relation $\rightarrow$ preserves niceness.*

Proof: For the first invariant, one notices that the only means of including an agent in a group are $init$, and fc. In all cases the recruited (perhaps created) agent is given a role and all its sites in $\mathcal{F}$ are set to some value. After that, no roles or logs are erased except via $init^*$, and $exit$, which also erase the group name.

For the second invariant, one sees that all rules but $init$, ps and $exit$ are about replacing simultaneously the (identical) logs at the two ends of an edge by a new log. So the invariant is maintained. The remaining rules $init$, ps and $exit$ do no set logs 2 or 1 on edges. (For the sake of this argument the root fictitious input is taken to be self-bound.) $\Box$

We suppose thereafter that all solutions are nice, so one can now say an edge has log 1 or 2, meaning both ends do, or equivalently one does.

Let g be a group name, define $T(g)$ (resp. $T_2(g)$), as the following subsolution of T:
- agents are those labelled with g in T
- among edges from the induced subgraph, only those set at 1 or 2 (resp. at 2) are kept.

Obviously $T_2(g) \subseteq T(g)$; one uses $T(g)$ to measure the progress of the recruitment signal, together with the partial embedding of the rule pattern, while one uses $T_2(g)$ to measure the progress of the local success signal back to the root.

We write $T \overset{g}{\rightarrow}^* T_1$ if the only group name used in the trace (a sequence of transitions) is g.

Lemma 4 *Suppose $[S] \overset{g}{\rightarrow}^*_{pre-ps} T$, then:*
- $T(g)$ embeds into $\alpha \cdot S$, the embedding being given by the roles;
- $T(g)$ is $\prec$ downward closed;
- $T_2(g)$ is $\prec$ upward closed.

Proof: A first thing to notice is that because T is nice, agents in $T(g)$ also have a role, and therefore the first clause makes sense. In the next two clauses it is understood that the closure is with respect to the order as inherited from the signal ordering $\prec$ defined on $\mathcal{F}$ via the embedding defined in the first clause.

We prove the three clauses by induction on the trace.

[8]Another information one can extract from the local confluence proof is that all sub relations defined by reducing further the type of rules allowed will still be confluent, since no commutation involved in the proof changes the type of rule under inspection.

Suppose the last rule is $init$, then it must be the first rule too, since the group name created is by definition unique. In which case the three clauses are trivially satisfied.

Suppose the last rule is an fc, then by definition of the rules, the bottom agent is given its role b (uniquely determined from $\mathcal{F}$), and the attendant edge $(a, i), (b, j)$ verifies $\mathcal{T}(a, i) = (b, j)$, and is set to 1, so the first clause follows. The second one holds too since all immediate $\prec$ predecessors are already at 1, and therefore by induction already in the partial embedding codomain. The third trivially does since no 2 were added.

Suppose the last rule is an lc, then by definition, the bottom agent has already role b and therefore is already in the embedding codomain (this is crucial as can be seen in the simple 'triangle' example in the next section), and the attendant edge $(a, i), (b, j)$ verifies $\mathcal{F} \setminus \mathcal{T}(a, i) = (b, j)$ and is set to 1, so the first clause follows. And so do the other two for the same reasons as above.

Finally, suppose the last rule is $resp$, the first two clauses are unchanged; the third one holds because the additional edge set at 2 has immediate successors already at 2. □

Here is now the principal lemma:

Lemma 5 *Suppose* $[S] \xrightarrow{g}{}^{*}_{pre-ps} T \xrightarrow{g}_{ps} T_1$*, then* $S \to S_1$*, with* $[S_1] = c(T_1)$.

Proof: Since ps applies, all outputs of the root are at 2.

Pick an agent with role a in $T(g)$, which is not the root; that agent must have been recruited by some fc rule (since this is the only way to enter a group while not being the root), at which time the edge to its principal input was set at 1; logs can only increase under $pre\text{-}ps$, so a's principal input edge is in $T(g)$. By down closure there is a path down to the root using only principal inputs and edges of log 1, or 2, and reaching to a root output at 2 (by assumption), and therefore by up closure, every site in that path has log 2 too, and so do the outputs of a (for the same reason). So every output site in $T(g)$ is at 2, and therefore so does every input, by niceness, and because by definition an output links to an input. And since $T_2(g)$ embeds in $\mathcal{F}$ (up to renaming), $T_2(g)$ can only be equal to $\mathcal{F}$, because $\mathcal{F}$ is connected.

So T is no other than $\alpha \cdot S = S_1$ up to logs (which are all at 2), group names and roles, and so is T_1. Now since $\to_c$ is confluent, one can choose to apply the remaining rules pp, and $exit$ in any order, and clearly there is a way to schedule those rewrites so that all logs are erased at the end, eg by using pp to switch all logs to 3, and only then use $exit$ for all agents. Hence $c(T_1) = [S_1]$. □

One sees that the last segment of the trace in fact entirely consists in $post\text{-}ps$ rules.

4.3 Bisimulation

Define the relation $S \approx T$ as $c(T) = [S]$. This is a functional relation from $m\kappa$ solutions to κ solutions. We are going to prove that it is a weak bisimulation, and conclude to the correctness of the compilation in the case of monotonic connected rules.

Lemma 6 $T \to^{*}_{post-ps} \to^{*}_{pre-ps} c(T)$.

Proof: It is enough to prove that any two successive transitions, the first in $pre\text{-}ps$, the second in $post\text{-}ps$ commute, which amounts to saying that two transitions one in $pre\text{-}ps^{*}$, and the second in

$post\text{-}ps$ are always commuting (concurrent in fact), which we know for a fact from the confluence proof of $\rightarrow_c$ above. $\Box$

Lemma 7 *Suppose $S \approx T$, and $T \xrightarrow{g}_{ps} T_1$, then $c(T_1) = S_1$, and $S \rightarrow S_1$ for some S_1.*

Proof: Suppose first there are some transitions of type $post\text{-}ps$ in $T \rightarrow_c^* [S]$. By the preceding lemma, one also has $T \rightarrow_{post-ps} T' \rightarrow_c^* S$. Now the transition $T \rightarrow_{post-ps} T$ is clearly concurrent to the ps one. Indeed the unique ps agent cannot match the unique $exit$ one (their input logs disagree), nor can it match the top pp one (same reason), nor the bottom pp one (because it cannot be the root). Therefore one has $T' \xrightarrow{g}_{ps} T_1'$, and $T_1 \rightarrow_{post-ps} T_1'$, for some T_1'. Now, because $\rightarrow_c$ is confluent, $c(T_1') = c(T_1)$, so piecing everything together, one has $S \approx T'$ (by assumption), and $T' \xrightarrow{g}_{ps} T_1'$, which (by induction) gives $c(T_1') = c(T_1) = S_1$, and $S \rightarrow S_1$.

Suppose now the first transition in $T \rightarrow_c [S]$ is of type $post\text{-}ps$, but does not involve agents in group g, then again this transition is concurrent with $T \xrightarrow{g}_{ps} T_1$. Indeed the ps agent cannot match any agent in the $post\text{-}ps^*$ rules, because all have a different group (by assumption), and one concludes as in the preceding case.

The remaining case to consider is when the trace $T \rightarrow_c [S]$ entirely consists in $pre\text{-}ps^*$ rules within group g, and there one can reverse all these, and apply the principal lemma to conclude. $\Box$

Here is the formal definition of bisimulation we are using.[9]

Definition 5 (Bisimulation) *Let $\{\rightarrow_1^{\ell_1}; \ell_1 \in L_1\}$, and $\{\rightarrow_2^{\ell_2}; \ell_2 \in L_2\}$, be LTSs with label sets L_1, L_2, and let f_1, f_2 be partial functions from L_1, L_2 to some set L, called* observations. *Observations extend naturally as total maps from L_1^*, L_2^* to L^*.*

Suppose γ_i is a trace (ie a sequence of successive transitions) in one of the above LTSs, we write $\ell(\gamma_i)$ in L_i^* for its sequence of labels.

A relation $\approx$ is an f_1, f_2-bisimulation if: whenever $S \approx T$, and $\gamma_1 : S \rightarrow_1^* S_1$, there exists $\gamma_2 : T \rightarrow_2^* T_1$ such that $f_2(\ell(\gamma_2)) = f_1(\ell(\gamma_1))$, and $S_1 \approx T_1$; and symmetrically.

Usually, one fixes the notion of observation by defining τ to be the only non observable label, and obtain the notion of weak bisimulation. Our variant definition allows for a little more flexibility, in that one can stipulate in an ad hoc fashion what it is that one observes in a given transition. This decision is embodied in the maps f_1, and f_2.

For our application, we consider on the κ side that all transitions are observed, so f_1 is a total map, and one observes which rule is being applied, that is to say $f_1(S, \alpha, \phi) = (S, \alpha)$. On the $m\kappa$ side, we take all transitions in $m\kappa$ to be 'silent', that is to say f_2 is undefined, unless the transition is of the ps type, in which case f_2 is defined and its value set to the κ rule (S, α) corresponding to the ps transition being applied.[10]

Theorem 1 *The relation $S \approx T$ defined as $c(T) = [S]$ is a weak bisimulation (wrt $\rightarrow_\kappa$, and $\rightarrow$).*

[9]The familiar cases are recovered as folows: set $L_1 = L_2 = L$, $f_1 = f_2 = Id$, for strong bisimulation; and if $\tau \in L$, set $f_1 = f_2 = Id$, except $f_1(\tau) = f_2(\tau) = \bot$, for weak bisimulation.

[10]This is to prepare the ground for a generalisation of the correctness statement to multiple rules, since we are dealing for the moment with only one κ rule. One could actually also observe the embedding defined by the role allocation, since by lemma 4 the two notions correspond.

Proof: Suppose $S \approx T$, $S \rightarrow_\kappa S_1$, and $T \rightarrow_c^* [S]$, then to respond to the challenge one simply simulates the κ reduction in $m\kappa$.

Suppose now one challenges the bisimulation on the $m\kappa$ side, and has $T \rightarrow T_1$. If that transition is not a ps, then it is either a $pre\text{-}ps$ one, which one can reverse, and so $c(T_1) = S$, or it is a $\rightarrow_c$ one, and by confluence, $c(T_1) = S$ too. In both cases, $c(T_1) = [S]$ and no response is needed on the κ side, since nothing is observed.

So the only interesting case is when $T \rightarrow_{ps} T_1$, where one applies the preceding lemma, to obtain an S_1 such that $[S_1] = c(T_1)$, and $S \rightarrow S_1$; and that obviously respects the observables as defined above. □

Since $S \approx [S]$ one obtains:

Corollary 2 *If* $[S] \rightarrow^* T$*, then* $S \rightarrow_\kappa^* S_1$ *with* $[S_1] = c(T)$*.*

5 Discussion

5.1 General rules

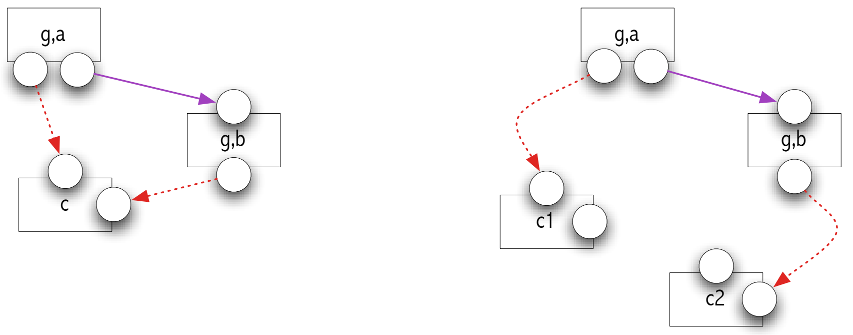
It is possible to construct a similar decomposition for a general κ rule that will not be monotonic and connected. Suppose first the rule of interest is antimonotonic and connected, then the compilation is in the same vein. One checks for the presence of the connected lhs during recruitement. Because and $fc2$, $fc3$, and $lc2$ are not needed, one uses only a subset of the rules above. And then one deals with the necessary erasures of edges and agents after the phase shift, where there is no possibility of failing.

To deal with general rules which are neither connected nor (anti-) monotonic, a simple solution is to add to the scenario enough statically defined edges to form a transitional connected component, and do the monotonic part of the rule during recruitment, and the anti-monotonic one (including the breaking of the transitional edges) after the phase shift. One needs enough sites to construct that fictitious super complex, so one also needs two additional sites for each agent at translation time (instead of just one as in the connected case).

As for how to deal with a set of rules, it is enough to give disjoint domains to distinct scenarios, so that one can tell from the role which rule is being attempted, and be sure in this way that there is no interference between those. The theorem above holds as is.

5.2 Deadlocks

As explained earlier, the use of a spanning tree $\mathcal{T}$ makes it possible to statically fix which agent will recruit a given one, while all the other agents have to use the later contact rules. Barring the use of reversible rules, some deadlocks of a special kind may happen if we do not do this. Below is an example of an atomic monotonic κ-rule, which is one of the simplest non binary ones since it has only three agents, and will give us an example of this particular kind of deadlock.



The rule is presented on the left and is attempting to construct a triangle; $\mathcal{F}$ is indicated by the edge orientation. Suppose now one suppresses the spanning tree constraint in the rules, and does no longer distinguish between a first and a later contact. It may happen that agents a, b attach to different cs, eg as shown on the right hand side. This creates a self-deadlock, since with the synchronisation provided by the spanning tree, which here boils down to choosing who from a and b should start binding a c, that could have been avoided.

The interesting thing here is that the cs will never receive a connection on their other inputs. So because one has introduced reversible rules, it may be that one can do without such a discipline. Note however that one would have to modify accordingly the invariant of lemma 4, which is clearly violated, since the map from the group to the rule rhs is no longer injective.

5.3 Mistakes

What one cannot do without, or so it seems, is the notion of group names. Suppose for a moment that one leaves group names out, then the rules could lead to outright mistakes. Let us take as an example a variant of the triangle rule above. In process notation:

$$a(l^x, r^y) + b(l, r^y) + c(l^x, r) \to a(l^x, r^y) + b(l^z, r^y) + c(l^x, r^z)$$

One sees in Fig. 4 that assuming one takes two copies of each agent in the suitable initial state, the situation can evolve into b not knowing which c it has to bind to, even in the presence of the spanning tree discipline. From there the situation can evolve in a completely symmetric way, that will lead to agreeing on an hexagon, instead of the intended two triangles.

Fig. 5 shows the result of running a stochastic version of the algorithm using ten of each type of agent. One sees an enneagon has been constructed, where all agents have a local view which is consistent with belonging to the triangle they should belong to.

This example leads to thinking it may be impossible to do without groups, and one can mount a general argument that κ cannot be compiled into binary κ. However one would have to spend quite some time explaining what is allowed in the process of compilation, and what not. For instance, the translation could introduce an enumeration of the initial solution (assigning unique identifiers) and obviously break any argument, since unique identifiers are as powerful as group names, at least in the absence of agent creation. So one would have to ask for compositional translations such that $[S + S'] = [S] + [S']$, where $+$ here means a disjoint sum. Another problem would be that one could

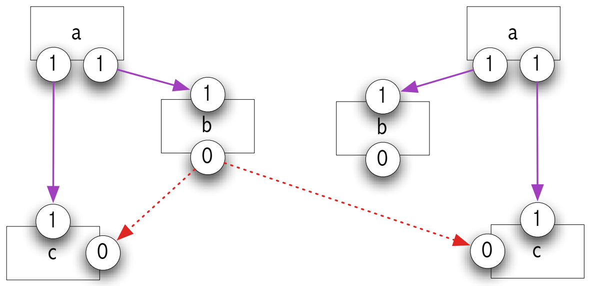


Figure 4: Without group names, lc is ambiguous, and b does not know to which c it should connect.

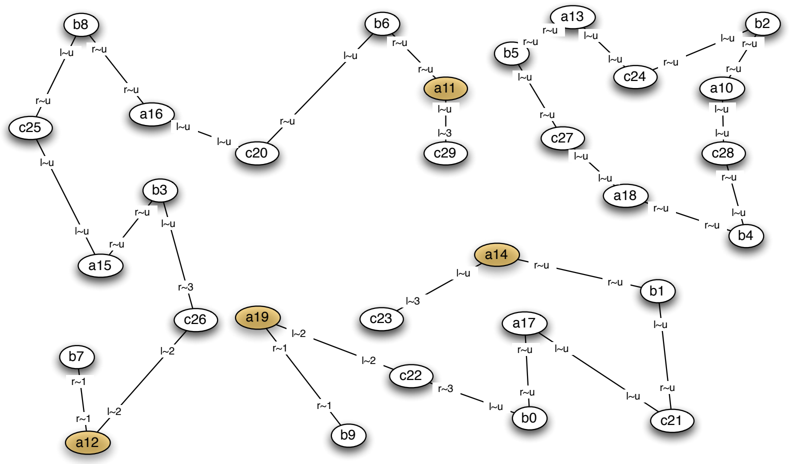


Figure 5: The ambiguity of the lc rules, in the absence of group names, is causing other shapes than the expected triangles to appear. The particular state presented above was obtained by a stochastic simulation starting with 10 of each agents. A 9-gone has formed already, but the the remaining chains are still active, and with probability will eventually loop. Note also that the only remaining active agents of type a are the chain ends.

encode κ connected components by quite arbitrary ones in $m\kappa$, so one could also decide only to admit translations which are perhaps identical up to new internal states, just as the one presented here is. At the moment, and for want of a clear cut notion of reasonable translation as one can find in ordinary process algebras [15], we will not venture into proving a negative result, and just develop an informal argument.

Consider a initial κ solution as in Fig. 4 (but without logs), and suppose now one uses a 'reasonable translation' of the associated rule. Certainly part of being a reasonable translation is the simulation property, so one can bring the left agents a, b, and c to a triangle. Say the obtained trace decomposes as $\gamma s \gamma'$ where s is the last step where the b, c edge is added (it could be added many times in principle). Suppose one takes a copy of that complete trace but applied on the remaining right agents, and runs the two copies concurrently, except that at the time of binding one swaps the two

cs. Since the binding rule is binary, and the two traces were identical and coming from isomorphic initial states, this will works as long as the symmetry is not broken by the very process of translation, a possibility which we would construe as not being a reasonable translation. Now post composing with the two copies of γ' is still possible, since the swap results in a set of local views which is wholly compatible with being in a triangle. So the remaining copies of γ' using only binary rules will not be able to tell the difference either, and obtain an hexagon, which breaks the bisimulation property.

Importantly, this informal swap argument while valid in particular for a tree-like rule pattern does not lead to breaking the bisimulation, since the swap does not produce any unintended rewrite in this case. And indeed one can encode everything binarily, as we saw earlier, since the later contact rules are not needed. The other nice thing about this particular case being that all there is to do to define a scenario is to choose the initiator, and hence there is no spanning tree concerns. In practice, it could well be that all biologically plausible rules are of this form.

6 Conclusion

We have presented a distributed implementation scheme for a certain class of graph rewriting rules using a specific agent-based language. The distribution is based on binary interactions, and the implementation seems quite natural, since the grain of distribution is chosen to be the node, that is to say every node is represented as an individual agent, and no auxiliary agent is introduced. Having said that, it seems difficult to say mathematically how good or natural the algorithm we proposed is.

To illustrate this delicate point, consider the following alternate self-assembly described in process terms. An agent wakes up and decides to apply some rule involving n other agents, he then sends n recruitment messages on a public channel; receiving agents send back a complete description of their internal state and bindings with their neighbours (using say a unique name for describing themselves, and therefore having to communicate with their neighbours to get their unique names too); when the initiator has collected all answers, he goes on to compose them together (uniquely since recruited agents uniquely identified themselves) into a complete description, and if that is by chance the lhs of the rule under consideration, he sends a success message to all waiting recruitees, or else a failure message. In the former case the needed actions are taken, in the latter, every agent exits the failed transaction. This informal description can easily be converted into a precise algorithm, say in π-calculus.

Now the problem is that no matter how unnatural and inefficient (this protocol will almost never succeed in a probabilistic world with large collection of agents, since it doesn't follow the extant bindings) this is, it it certainly correct in the sense ours is correct. Perhaps a dividing line is that in our self-assembly, no agent knows which rule is being applied, whereas in the example above, the initiator has to know everything about the rule. Note that, as said earlier, a consequence of our analysis is that no name creation is needed for tree-like patterns, whereas the 'star' algorithm described above needs one for each agent. Still it remains to see how one can give a mathematically grounded and robust notion that would clearly separate those two compilations, and whether the informal case made for the necessity of group names, or for any equivalent mechanism, can be made a bona fide argument.

It remains to discuss briefly how such a study may matter from a biological point of view.

For one thing it is always good to study as we did the abstract mathematical properties of a modelling language, to better understand it, and develop well-grounded environments for modelling. Before one want to confront the questions of how true or even plausible a model of a pathway is (there is an estimated four hundred BNG rules needed to model the EGF pathway at the level of knowledge one has today), and how far one can rely on it to predict correct behaviours under various perturbations, there is the earthly question of how does one write down such a model in the first place. There is need for an ad hoc software engineering, and surely a study probing in the computational limits of the language makes one wiser in this respect. To be more ambitious and say that this computational enquiry can directly help in understanding the intended systems, one would have to believe, as we do, that there is a meaningful qualitative (ie non kinetic) way of studying pathways.

References

[1] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. *BioSystems*, 83:136–151, January 2006.

[2] M.L. Blinov, J.R. Faeder, and W.S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20:3289–3292, 2004.

[3] Vincent Danos, Jean Krivine, and Pawel Sobocinski. General reversibility. In *EXPRESS'06*, ENTCS. Elsevier, July 2006. To appear.

[4] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, September 2004.

[5] J.R. Faeder, M.L. Blinov, Goldstein B., and W.S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Complexity*, 10:22–41, 2005.

[6] J.R. Faeder, M.L. Blinov, and W.S. Hlavacek. Graphical rule-based representation of signal-transduction networks. *Proc. ACM Symp. Appl. Computing*, pages 133–140, 2005.

[7] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comp. Phys.*, 22:403–434, 1976.

[8] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem*, 81:2340–2361, 1977.

[9] W.S. Hlavacek, J.R. Faeder, M.L. Blinov, R.G. Posner, M. Hucka, and W. Fontana. Rules for Modeling Signal-Transduction Systems. *Science's STKE*, 2006(344), 2006.

[10] Kurt W. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular Biology of the Cell*, n. 10:2703–2734, 1999.

[11] Kurt W. Kohn and Mirit I. Aladjem. Circuit diagrams for biological networks. *Molecular Systems Biology*, January 2006.

[12] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile process (I and II). *Information and Computation*, 100:1–77, 1992.

[13] Kanae Oda and Hiroaki Kitano. A comprehensive map of the toll-like receptor signaling network. *Molecular Systems Biology*, 2, April 2006.

[14] Kanae Oda, Yukiko Matsuoka, Akira Funahashi, and Hiroaki Kitano. A comprehensive pathway map of epidermal growth factor receptor signaling. *Molecular Systems Biology*, 1, May 2005.

[15] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In *POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 256–265, New York, NY, USA, 1997. ACM Press.

[16] Andrew Phillips, Luca Cardelli, and Giuseppe Castagna. A graphical representation for biological processes in the stochastic pi-calculus. *Transactions on Computational Systems Biology*, 4230(7):123–152, 2006.

[17] Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 2001.

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

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

[20] Glynn Winskel. Event structure semantics for ccs and related languages. In *Proceedings of 9th ICALP*, volume 140 of *Springer*, pages 561–576. LNCS, 1982.