Computational Self-assembly

Theoretical Computer Science 404 (2008) 61–75

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Computational self-assembly

Pierre-Louis Curien [a], Vincent Danos [a,b,*], Jean Krivine [c], Min Zhang [d]

[a] CNRS & Université Paris-Diderot, France
[b] University of Edinburgh, United Kingdom
[c] École Polytechnique, France
[d] East China Normal University, China

A B S T R A C T

The object of this paper is to probe the computational limits of an applied concurrent language called κ. This language describes how agents can bind and modify each other. It is meant as a syntactic medium to build, discuss and execute descriptions of cellular signalling pathways. However, it can be studied independently of its intended interpretation, and this is what we are doing here.

Specifically, we define a reduction of κ to a fragment where interactions can involve *at most two agents* at a time. The translation relies on an implicit causality analysis which permits escaping deadlocks. It incurs only a linear blow up in the number of rules. Its correctness is spelt out in terms of the existence of a specific weak bisimulation and is proved in detail. To compensate for the binary restriction, one allows components to create unique names. When using acyclic rules, this additional facility of name creation is not needed and κ can be reduced to a binary form as is.

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 and revision [1,2]. A similar project by Oda, Kitano and collaborators provides an extensive account of the EGF receptor pathway [3].

The κ language was introduced a few years ago (and named κ as a reference to Kohn maps) in another attempt to give means to express conveniently the combinatorial detail at hand and support numerical simulation [4–7]. A similar language, BNG, was proposed independently [8–11]. The current κ implementation incorporates a natural extension of Gillespie's algorithm [12,13] that given kinetic rates turn the non-deterministic transition system associated with a 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). The simulation algorithm shows good complexity properties and, in particular, is insensitive to the combinatorial explosion of species that one often finds in signalling models [14].

Because κ is rule-based 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 [7]. Even a simple EGF model (see Section 3 for a brief presentation) generates hundreds of different complexes and thousands of flat reactions. Another valuable consequence of dealing with structured agents is that one can use the attendant notion of causality (when one event appears necessarily before another one in a given simulation) to simplify the representation of simulation trajectories. This

* Corresponding author at: CNRS & Université Paris-Diderot, France.
E-mail address: vincent.danos@gmail.com (V. Danos).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.04.014

leads to informative representations of simulations as partial-time minimal configurations, which, in turn, help in controlling the key kinetic junctures of a model [6].

In the present paper we leave aside such concrete modelling issues and study κ for its own sake, as a computational framework based on graph-rewriting. We specifically ask whether the ability of rewriting any number of agents at once can be dispensed with, and if 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. Our result does not deal with the engineering aspects which are traditionally an integral part of the problem. To borrow from the language of synthetic biology, we suppose here that we are working with idealised and well specified parts.

We solve the question in the positive, although it is important to realise that one cannot simply binarily restrict κ. In order to recover full expressiveness, one has to endow the binary restriction 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. We also show that rules using no cyclic complexes can be interpreted without the additional name creation facility. Finally we offer an informal argument showing that it would probably be very difficult to do without name creation in the general case. This kind of negative result is notoriously tricky to set up, and we keep it informal and inconclusive, though we find it persuasive.

Our contribution can be set in different perspectives.

From an algorithmic point of view it includes a terse and efficient distributed implementation of a class of graph-rewriting systems, with the natural granularity choice of having agents be the graph nodes. Rewrite rules are broken down between agents (none of the agents actually know the rules) and the application of a global rule needs the setting up of a distributed consensus by means of binary asynchronous interactions. The actual solution uses reversible rules in order to escape deadlocks, and this is done by keeping a careful track of causality in a way that follows broadly the 'history category construction' [15,16]. The proof is done in detail, which is important for such rather large and non intuitive agent systems.

From a process or agent-based language point of view,[1] this is a way of measuring the computational power of κ, and one can easily derive a compilation down to π-calculus [17] from there [4]. This relates to early work using directly π-calculus to model pathways [18–21], in that it gives a structured and systematic way of obtaining such lower-level descriptions (as a compilation should) from a higher-level notation.

But, more importantly, it also says that there is a natural and naturally distributed task for which π-calculus is complete. 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. The rewriting framework

We start with a brief description of κ.

A partial matching on a set X is a partial involution on X that has no fixed point.

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

A κ-*solution* S is a set of agents, together with a partial matching ν on the set $\sum_{a\in S} \sigma(a)$ of all sites in S. A site (a, i) (not) in the domain of ν is said to be bound (free). Since ν is a matching no site can be bound twice.

Every solution has an underlying graph structure, and we shall freely use graph-theoretic notions such as subgraph or connected component. 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 state is preserved
$\nu(a, i) = (b, j) \Rightarrow \nu'(\phi(a), i) = (\phi(b), j)$	edges are preserved
$\nu(a, i) = \perp \Rightarrow \nu'(\phi(a), i) = \perp$	free sites are preserved.

The last condition is somewhat unusual but important as it allows one to specify as a condition for applying a rule (see below) that a given site is free. The use of sites to control embeddings make the notion stronger than that of usual graph morphisms. In particular, a non-empty partial map from a connected S to some S' extends to at most one embedding.

[1] By agent-based language, one usually means a language defining processes offering interactions with other processes, conditioned on the participants' states, and resulting in redefining states and subsequent interaction offers for each participant. 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 $\Sigma : \mathcal{A} \to \wp(\mathcal{S})$, 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 is fixed once and for all.

Atomic actions one wishes to perform on a solution 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 agent may share.

An *action* α 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 left hand side is written $\alpha \cdot S$ (the rule 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 $\phi : S \to S'$, 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. Note that since S may have symmetries, specifying only the codomain of ϕ does not in general suffice to determine the result. 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.

Definition 1. A set R of rules defines a labelled transition relation:

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

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

When rules are given rates, the above transition system can be enriched as a continuous time Markov chain [14]. This generalizes the traditional stochastic structure associated to Petri nets (aka multiset rewriting systems or coupled chemical reactions) [13]. Although it would be very interesting to examine the quantitative aspects of our self-assembly result, we shall not use quantitative semantics in the present study.

3. An example

To get a sense of how κ can be used in a modelling context we give a short example.

A set of (atomic) rules for the early response to EGF (epidermal growth factor) is shown Fig. 1. The graphical notation represents actions as dotted lines. The actual signalling network comprehends about ten signals and four different EGF receptors [3], so the model is really a simplification of the pathway, but the principle stays the same: binding, unbinding and modification events can be directly represented; one can then tailor the left hand side of a rule to specify under which conditions such an event may happen; eventually one can further refine and calibrate the system's behaviour by adjusting rates. For instance, the binding of Shc to EGFR (Fig. 1, left column, last rule) is conditional on the Y1148 site being in a certain internal state. A more detailed and comprehensive version of this rule set can be found in Ref. [6] together with considerations on its quantitative behaviour and how it relates to the causal structures implicit in the rule set.

Already with this simple set of rules, one sees that many non-isomorphic complexes can be constructed. One example of such a complex is shown Fig. 2; it is constructible using sufficiently many unphosphorylated and disconnected agents of the appropriate types (a reasonable initial state for a signalling network).

This short example is also the occasion to introduce an in-text process notation which is an alternative to the graphical notation. It is sometimes convenient, and closer to the implementation. Using this notation the first two rules in the left column of Fig. 1 can be written:

$$\begin{aligned}
&\mathrm{EGFR}(l, r), \mathrm{EGF}(r) \to \mathrm{EGFR}(l^x, r), \mathrm{EGF}(r^x)\\
&\mathrm{EGFR}(l^{x_1}, r^y, Y1148_u), \mathrm{EGF}(r^{x_1}), \mathrm{EGFR}(l^{x_2}, r^y), \mathrm{EGF}(r^{x_2}) \to\\
&\mathrm{EGFR}(l^{x_1}, r^y, Y1148_p), \mathrm{EGF}(r^{x_1}), \mathrm{EGFR}(l^{x_2}, r^y), \mathrm{EGF}(r^{x_2}).
\end{aligned}$$

Bindings (internal states) are indicated as superscripts (subscripts) to their sites, the rule pattern is the left hand side, and the rule action can be deduced by comparing the right and left hand sides. For instance in the first rule, the action consists in creating an edge, represented here by x shared as a superscript by sites l in EGFR, and r in EGF.

Because of potential symmetries in the left hand side, this new notation only carries enough information to define the semantics if, in addition, a partial injection from the left hand side agents to the right hand side ones is given. Usually it is clear from the context which mapping is meant. On the other hand the graphical notation is unambiguous ion itself, making it easy to enrich rules with biologically meaningful information such as the EGFR activating site being responsible for modifying the internal state associated to EGFR(Y1148) and Shc(Y317) (Fig. 1, left column, second and third rule). That information is not needed for the semantics though.

With the rewriting framework and a few examples in place we will turn to the specific question we are interested in, namely the self-assembly problem. However 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.

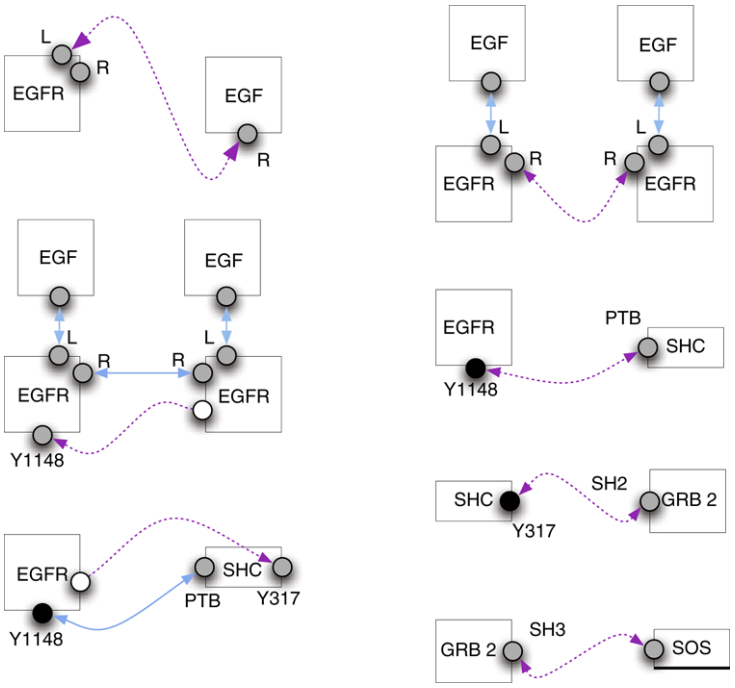


Fig. 1. A sample of rules modelling early EGF signalling: actions are represented as dotted lines; the left column includes signal/receptor binding, receptor activation, and Shc activation; the right column shows receptor dimerisation, Shc binding, binding of Grb2, and binding of SoS; in the two modification rules the EGFR site responsible for the various phosphorylations is represented as a solid white circle.

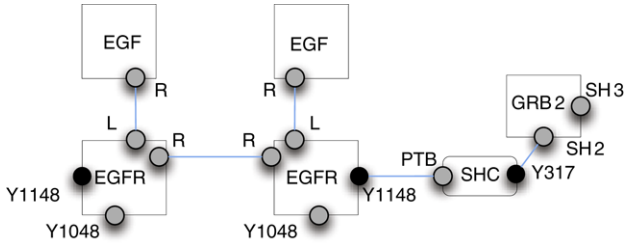


Fig. 2. An example of a complex (connected solution); nodes correspond to proteins, sites correspond to protein domains or/and amino-acid residues susceptible of post-translational modifications (solid black).

4. Turing machines

Suppose given a Turing machine, that is to say, an alphabet M, a set of states Q, and a transition function δ from $M \times Q$ to $M \times Q \times \{\leftarrow, \rightarrow\}$. To represent a tape element, we need only use one agent type, which we can therefore keep nameless. Agents of this type have each a 'left' and a 'right' binding site written l, and r, as well as '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 $*$ to indicate a passive tape element (not under the head), and q both as a representation of the current state and to indicate an active tape element. The intention is that only one tape element is active at a time (the one the 'head' currently points at).

Left (see also Fig. 3) and right transitions $\delta(a, q) = (b, q', \leftarrow)$ or $(b, q', \rightarrow)$ translate in binary non atomic rules:

$$\langle u_*, r^x\rangle, \langle l^x, d_a, u_q\rangle \rightarrow \langle u_{q'}, r^x\rangle, \langle l^x, d_b, u_*\rangle \quad \text{left transition}$$
$$\langle d_a, u_q, r^x\rangle, \langle l^x, u_*\rangle \rightarrow \langle d_b, u_*, r^x\rangle, \langle l^x, u_{q'}\rangle \quad \text{right transition.}$$

One also needs rules to handle unbounded computations, providing means to extend the tape on both sides, and to produce more blank tape elements.

$$\langle l, r\rangle, \langle l, u_q\rangle \rightarrow \langle l, r^x\rangle, \langle l^x, u_q\rangle \quad \text{left tape extension}$$
$$\langle u_q, r\rangle, \langle l, r\rangle \rightarrow \langle u_q, r^x\rangle, \langle l^x, r\rangle \quad \text{right tape extension}$$
$$\rightarrow \langle l, u_*, d_B, r\rangle \quad \text{tape element creation.}$$

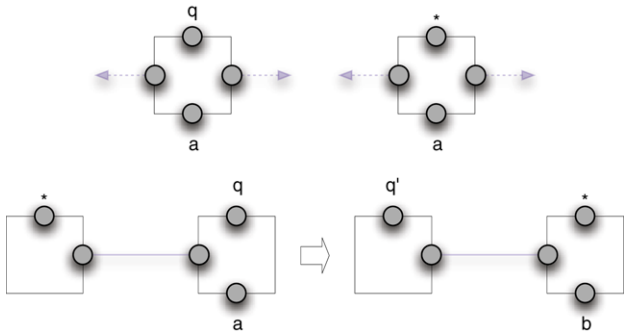


Fig. 3. Tape elements have four sites: the upper site is used to store the machine state, when the tape element is active (top row, left agent), and when it is not one uses $*$ instead (top row, right agent); the lower site holds a symbol; the left and right sites are used to chain together tape elements. Left transition rule (bottom row): only internal states are modified, and the rule is not atomic since it modifies at least two internal states.

Both tape extension rules test for the presence of the activity token q so that one only attaches new tape elements to the current tape, that is to say, the one that holds q. One also tests that both sites l and r of the new element are free in order to prevent the formation of cyclic tapes which one could obtain by tying both ends.

Those five rules preserve the fact that there is a unique connected component holding a unique q value (the tape), which is also the only one with possibly more than two agents. This obtains a simple encoding. One can further observe that the non atomicity of the transition rules violates the strong locality conditions on rule sets given in Ref. [5] and that is not a coincidence, since such rule sets cannot preserve global invariants and in our specific case would inevitably lead to cyclic tapes.

One could consider variants: the last three rules could be combined with the upper two to create on demand a left (right) neighbour to move the head to. In this case the encoding halts effectively when the machine does, else new tape elements can still be produced by the creation rule and perhaps attached to the tape, although the head (that is the activity token) will no longer move. The encoding is suggestive of other non sequential machines, eg one could have more than one active agent on the tape, then the system would behave as an asynchronous multi-headed TM, with heads being on the same tape.

This simple encoding of Turing machines uses no further resources than those of binary κ. By contrast, to provide an encoding of κ seems to require a significant enrichment of the agents range of behaviour as we shall see next.

5. 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 at once. For instance, four agents are needed in the EGFR dimerisation rule shown above (Fig. 1, right column, first rule). Could one obtain the same behaviour by using only binary or unary rules? Or to rephrase the question in a more intuitive fashion, is there a principled way for agents to build enough awareness of their surroundings so that all interactions can become local? This is not unlike asking how one decomposes a complex chemical reaction into elementary ones, which is something one usually does for the sake of assigning kinetics to that complex reaction. This notion of communication expressiveness we are now looking for should not be confused with sequential expressiveness in the sense of Turing machines, which we have seen in the preceding section.

5.1. Enrichment

Suppose given a countable set of group names, one now allows agents to use one of these group names in their internal state. In addition a rule action can now incorporate the creation of a new group name. That is the key and only additional computational feature we will use here. In particular no computation can be done on group names, further than comparing them for equality (as in π-calculus). Whether this enrichment is truly necessary is an interesting question to which we will return in the last section.

The resulting variant of κ where rules and agents can create and compare group names, but rules are restricted to be at most binary, will be called $m\kappa$.

In graphical notation group names will be represented as additional labels to the agent.

5.2. The idea

Before we describe the compilation of κ into $m\kappa$ in detail let us explain it informally. It has the pleasing property that an agent is sent to an agent (with a slightly larger internal state space) and no auxiliary agent is introduced. Thus it preserves the natural granularity of distribution, that is to say nodes are independent processes. Since rules with more than two agents are forbidden, such rules will have to be broken down into smaller ones. The idea is to replace the instantaneous recognition of the rule pattern afforded by arbitrary rules, and materialised by the embedding ϕ in (1), by a gradual and distributed

construction of such an embedding which will involve many (binary or unary) smaller rules. To this effect, a group name is created and propagated to build a transient cooperative structure corresponding to a partial embedding, partially acted on by the rule action, 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.

5.3. Scenarios

To organise the construction of that partial embedding one uses a statically defined structure called a *scenario* which depends on the rule of interest.

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 $\mathcal{F}$
- $in \in S$ is the common root of $\mathcal{F}$ and $\mathcal{T}$.

Such scenarios always exist, since the rule being monotonic and connected, the right hand side $\alpha \cdot S$ must be connected, and any connected graph admits an acyclic orientation. This can be obtained, for instance, by choosing any node as a 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 $\alpha \cdot S$).

Note that a scenario is not defined in a unique way, there may be many for the same rule.

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 all this later.

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

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

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

Saying that $\mathcal{F}$ extends $\mathcal{T}$ amounts to saying that $\mathcal{T}(a,i) \neq \bot \Rightarrow \mathcal{T}(a,i) = \mathcal{F}(a,i)$.

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 $\mathcal{F}(a,i) \neq \bot$, an input if $\mathcal{F}^*(a,i) \neq \bot$, and a principal input if $\mathcal{T}^*(a,i) \neq \bot$.

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 be at least one site for each agent (because $\mathcal{F}$ is connected).

We will use the following $\mathcal{F}$-induced 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:

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

5.4. The rules

We suppose now we are given a monotonic connected rule and a scenario $\mathcal{F}$ for the connected and monotonic rule S, α, and define the associated set of $m\kappa$ rules.

To make notations less daunting, we suppose 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 easily extend to those cases since they are purely local to an agent.

The rules can be organised in various subgroups. 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 was a suitable embedding for S, and also knows that the action 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 as noted before), 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* (Section 5.4.1)
- the recruitment rules where signal flows down $\mathcal{T}$: *fc* (Section 5.4.2)
- the secondary recruitment rules where signal flows down $\mathcal{F} \setminus \mathcal{T}$: *lc* (Section 5.4.3)
- the response rules where signal flows back up $\mathcal{F}$: *resp* (Section 5.4.4)
- the unique phase-shift rule: *ps* (Section 5.4.1)
- the post phase shift propagation rules: *pp* (Section 5.4.4), and *exit* (Section 5.4.1).

We now carefully review a number of conventions used in the graphics, as well as some terminology which is convenient when discussing rules.

The $m\kappa$ rules are represented in the left/right hand side manner and using a graphical style. This is better suited to handling several modifications at each step. Some rules are reversible and this is indicated by the use of double sided arrows. Group names are written g, other names a, b, stand for agents of $\mathcal{F}$, and are referred to as *roles*. For any agent taking part in the rules below, the sites that are represented in those rules are those occurring (bound) in $\mathcal{F}$; those sites may bear an internal state which is an integer ≤ 3 referred to as a *log*. 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 represent 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 the initiator agent *in* are set to 0.

5.4.1. 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.

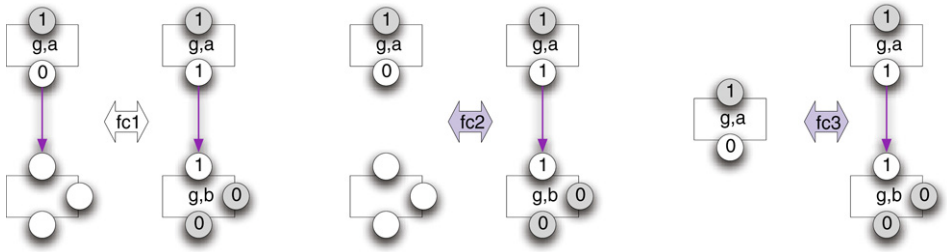
The middle rule is the phase shift. The root has collected all success signals (all the outputs are at 2) and shifts to the next phase; success is guaranteed here so the rule is not reversible.

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 $\mathcal{F}$.

5.4.2. 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.

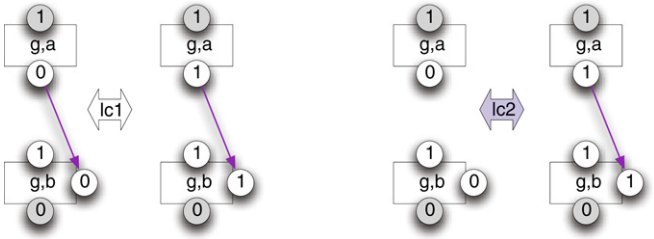
In all three cases, all of the top agent inputs have to be at 1, and the (a, i) output log is set to 1, the newly recruited bottom agent has its principal input log set to 1, while its other logs (secondary inputs and outputs) are all set to 0. The bottom agent is also labelled by its role b in $\mathcal{F}$,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



5.4.3. 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.

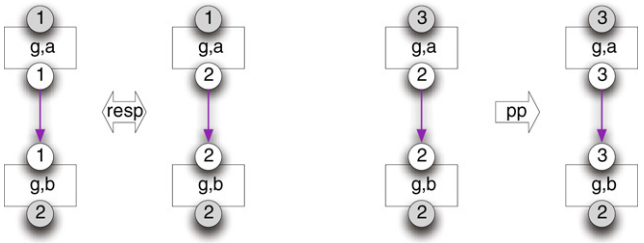


5.4.4. Responses, and propagation

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 a 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 cannot 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.



5.5. Discussion

All the forward and backward rules above necessarily involve at most two agents. 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 the 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.

6. Correctness of self-assemby

We now discuss various mathematical properties of our self-assembly translation, 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 a site to hold the group name and role during exploration, and a fictitious site to be used in the *init* rule.

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

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

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

Note that 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.

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-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).

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.

6.1. The relation $\to_c$ is confluent

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

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

Proof. Let T be a solution, with n_i occurrences of $\log i$ ($i = 0, 1, 2, 3$), and set $\rho(T) = p_0n_0 + p_1n_1 + p_2n_2 + p_3n_3$, 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 $T \to_c T'$ then $\rho(T') < \rho(T)$, and strong normalization follows. □

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

Proof. Suppose $T \to_c T_1$, and $T \to_c T_2$. 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 only the bottom agent 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 $\to_c$ 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 $\to_c$ 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 $\to_c$ 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 from embedding 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 $\to_c$ (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.[2]
- 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.[3]

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

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

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 $\to_c$ reductions to normal form have same length.[4]

6.2. The principal lemma

Let us 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 $\to$ 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 not set logs 2 or 1 on edges. (For the sake of this argument the root fictitious input is taken to be self-bound.) □

---

[2] 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 backwards, 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.

[3] 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. [15].

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

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 sub-solution 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 $T \xrightarrow{g}{}^{*} T_1$ if the only group name used in the trace (a sequence of transitions) is g.

Lemma 4. *Suppose* $[S] \xrightarrow{g}{}^{*}_{pre\text{-}ps} T$, *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. The 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.

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. 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* $[S] \xrightarrow{g}{}^{*}_{pre\text{-}ps} T \xrightarrow{g}_{ps} T_1$, *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-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 is every input, by niceness, and because by definition an output links to an input. 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 none 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-ps* rules.

6.3. Bisimulation

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

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

Proof. It is enough to prove that any two successive transitions, the first in *pre-ps**, the second in *post-ps* commute, or equivalently that two transitions one in *pre-ps*, and the second in *post-ps* are always commuting (concurrent in fact), which is proven as in the (local) confluence proof of $\to_c$ above. □

Lemma 7. *Suppose* $S \approx T$, *and* $T \xrightarrow{g}_{ps} T_1$, *then* $S \to S_1$, *with* $[S_1] = c(T_1)$.

Proof. The proof is by induction on $T \to_c^* [S]$, and inspection of the first transition in that trace.

Suppose $T \to_{post\text{-}ps} T' \to_c^* [S]$, then the transition $T \to_{post\text{-}ps} T'$ is concurrent to the *ps* one: 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 that bottom agent cannot be the root). Hence, on the one hand, $T' \xrightarrow{g}_{ps} T'_1$, and $T_1 \to_{post-ps} T'_1$, for some T'_1, and on the other hand (because $\to_c$ is confluent) $c(T'_1) = c(T_1)$, so by induction $c(T'_1) = c(T_1) = [S_1]$, and $S \to S_1$.

Suppose $T \to_{pre\text{-}ps^*} T' \to_c^* [S]$, and no agent in g is involved in $T \to T'$, then again this transition is concurrent with $T \xrightarrow{g}_{ps} T_1$: the *ps* agent cannot match any agent in the *pre-ps** rules, because all have a different group (by assumption), and one concludes as in the preceding case.

By the preceding lemma, the only remaining case to consider is when the trace $T \to_c^* [S]$ entirely consists in *pre-ps** rules within group g, and there one can reverse all these, and apply the principal lemma to conclude. □

Here is the formal definition of bisimulation we are using.[5]

Definition 5 (*Bisimulation*). Let $\{\to_1^{\ell_1}; \ell_1 \in L_1\}$, and $\{\to_2^{\ell_2}; \ell_2 \in L_2\}$, 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 $\gamma_1 : S \to_1^* S_1$, there exists $\gamma_2 : T \to_2^* T_1$ 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 obtains 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.[6]

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

Proof. Suppose $S \approx T$, $S \to_\kappa S_1$, and $T \to_c^* [S]$, 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 $T \to T_1$. If that transition is not a *ps*, then it is either a *pre-ps* one, which one can reverse, and so $c(T_1) = S$, or it is a $\to_c$ 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 $T \to_{ps} T_1$, where one applies the preceding lemma, to obtain an S_1 such that $[S_1] = c(T_1)$, and $S \to S_1$; and that obviously respects the observables as defined above. □

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

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

7. Discussion

7.1. 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 recruitment. Because *fc2*, *fc3*, and *lc2* are not needed, one uses only a subset of the rules above, 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] 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.

[6] 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.

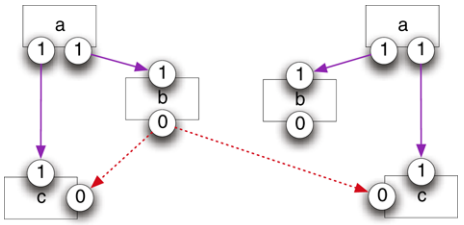
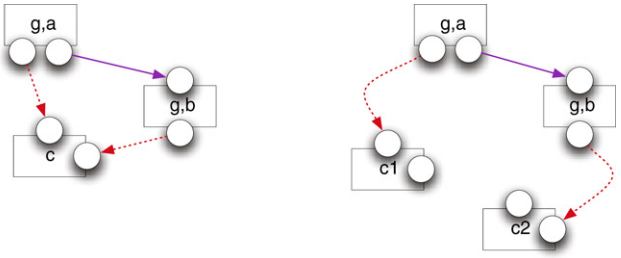


Fig. 4. Without group names, *lc* is ambiguous, and *b* does not know to which *c* it should connect.

7.2. 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 no longer distinguishes between a first and a later contact. It may then happen that agents *a*, *b* attach to different *c*s, 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 *c*s 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 and obtain a more asynchronous form of assembly. 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.

7.3. 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 that it may be impossible to do without groups, or an equivalent enrichment of κ. However one would have to explain what are the constraints bearing on compilation. 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 $[S, S'] = [S], [S']$. Another problem would be that one could 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, we will not venture into proving a negative result, but 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

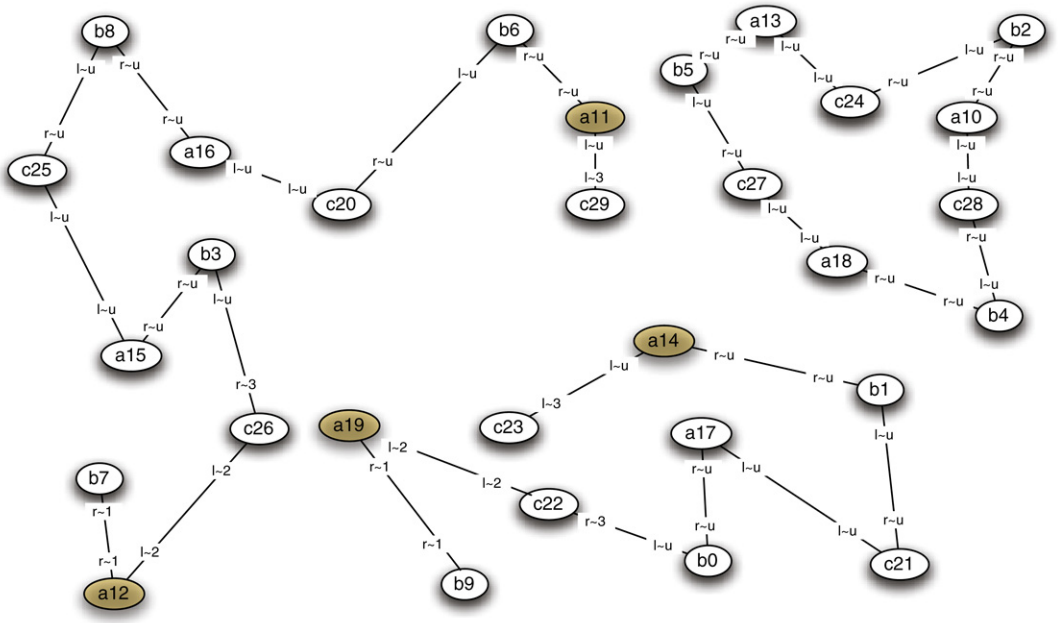


Fig. 5. The ambiguity of the *lc* rules, in the absence of group names, is causing other shapes than the expected triangles to appear. The particular state presented above was obtained by a stochastic simulation starting with 10 of each agents. A 9-gone has formed already, but the the remaining chains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

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 *c*s. Since the binding rule is binary, and the two traces were identical and coming from isomorphic initial states, this will work 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. 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 concern. It could be that a fair number of plausible signalling rules are of this form.

8. Conclusion

We have presented a distributed implementation scheme for a certain class of graph rewriting rules using a simple agent-based language of binding and modifications. 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 scheme (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 composes them together (uniquely since recruited agents uniquely identified themselves) into a complete description, and if that is by (extraordinary) chance the left hand side 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 a large collection of agents, since it doesn't follow the extant bindings) this is, it is certainly correct in the sense we have proved ours is. What then is the difference? A clear 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

rather dumb algorithm described above needs one for each agent. It remains to be seen 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 into a bona fide argument.

To conclude, let us 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 wants to confront the questions of how true or even plausible a model of a pathway is (we have at the moment about four hundred rules describing the EGF pathway at the best resolution), 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 some ad hoc software engineering, and surely a study probing in the computational limits of the language makes one wiser in this respect. More ambitiously one can imagine that this computational enquiry can directly help in understanding the intended systems by educating the eye of the modeller into recognising higher level synchronisation mechanisms encoded into systems of local interactions. Indeed, we have given one general strategy to do so. Our result could also be used for synthetic protein-protein interaction networks, allowing one to design global transformations with local ones that are presumably easier to engineer.

References

[1] K.W. Kohn, Molecular interaction map of the mammalian cell cycle control and dna repair systems, Molecular Biol. Cell 10 (8) (1999) 703–2734.
[2] Kurt W. Kohn, Mirit I. Aladjem, Circuit diagrams for biological networks, Molecul. Syst. Biol. (2006).
[3] Kanae Oda, Yukiko Matsuoka, Akira Funahashi, Hiroaki Kitano, A comprehensive pathway map of epidermal growth factor receptor signaling, Molecul. Syst. Biol. 1 (2005).
[4] Vincent Danos, Cosimo Laneve, Formal molecular biology, Theoret. Comput. Sci. 325 (1) (2004) 69–110.
[5] Vincent Danos, Jérôme Feret, Walter Fontana, Jean Krivine, Abstract interpretation of cellular signalling networks, in: F. Logozzo, et al. (Eds.), VMCAI'08, in: LNCS, vol. 4905, Springer, 2008, pp. 83–97.
[6] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, Jean Krivine, Rule-based modelling of cellular signalling, in: Luis Caires, Vasco Vasconcelos (Eds.), Proceedings of the 18th International Conference on Concurrency Theory, CONCUR'07, in: Lecture Notes in Computer Science, 2007.
[7] Vincent Danos, Agile modelling of cellular signalling, in: Proceedings of ICCMSE'07, 2007.
[8] W.S. Hlavacek, J.R. Faeder, M.L. Blinov, R.G. Posner, M. Hucka, W. Fontana, Rules for modeling signal-transduction systems, Science's STKE 2006 (344) (2006).
[9] J.R. Faeder, M.L. Blinov, W.S. Hlavacek, Graphical rule-based representation of signal-transduction networks, Proc. ACM Symp. Appl. Comput. (2005) 133–140.
[10] James R. Faeder, Michael L. Blinov, Byron Goldstein, William S. Hlavacek, Combinatorial complexity and dynamical restriction of network flows in signal transduction, Syst. Biol. 2 (1) (2005) 5–15.
[11] Michael L. Blinov, James R. Faeder, Byron Goldstein, William S. Hlavacek, A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity, BioSystems 83 (2006) 136–151.
[12] Daniel T. Gillespie, Exact stochastic simulation of coupled chemical reactions, J. Phys. Chem 81 (1977) 2340–2361.
[13] Daniel T. Gillespie, A general method for numerically simulating the stochastic time evolution of coupled chemical reactions, J. Comput. Phys. 22 (1976) 403–434.
[14] Vincent Danos, Jérôme Feret, Walter Fontana, Jean Krivine, Scalable simulation of cellular signaling networks, in: Z. Shao (Ed.), in: Proceedings of APLAS 2007, vol. 4807, 2007, pp. 139–157.
[15] Vincent Danos, Jean Krivine, Pawel Sobocinski, General reversibility, in: Proceedings of the 13th International Workshop on Expressiveness in Concurrency, EXPRESS 2006, in: ENTCS, vol. 175, Elsevier, 2007, pp. 75–86.
[16] P. Sobociński, Reversing graph transformations, in: Workshop on Petri Nets and Graph Transformations, PNGT '06, in: Electronic Communications of the EASST, vol. 2 (2006), 2006.
[17] Robin Milner, Joachim Parrow, David Walker, A calculus of mobile process (I and II), Inform. Comput. 100 (1992) 1–77.
[18] 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.), in: Pacific Symposium on Biocomputing, vol. 6, World Scientific Press, Singapore, 2001, pp. 459–470.
[19] Corrado Priami, Aviv Regev, Ehud Shapiro, William Silverman, Application of a stochastic name-passing calculus to representation and simulation of molecular processes, Inform. Process. Lett. (2001).
[20] A. Regev, E. Shapiro, Cells as computation, Nature 419 (2002).
[21] Andrew Phillips, Luca Cardelli, Giuseppe Castagna, A graphical representation for biological processes in the stochastic pi-calculus, Trans. Comput. Syst. Biol. 4230 (7) (2006) 123–152.