

Self-assembling Graphs

Vincent Danos (vincent.danos@pps.jussieu.fr)
Équipe PPS, CNRS & Université Paris VII

Fabien Tarissan (fabien.tarissan@pps.jussieu.fr)
Équipe PPS, Université Paris VII

March 13, 2006

Abstract. A self-assembly algorithm for synchronising agents and have them arrange according to a particular graph is given. This algorithm, expressed using an ad hoc rule-based process algebra, extends Klavins' original proposal [1], in that it relies only on point-to-point communication, and can deal with any assembly graph whereas Klavins' method dealt only with trees.

1. Introduction

In a number of different subject areas, nanotechnologies [2], amorphous computations [3], molecular biology [4], one commonly finds a debate about whether and how complex shapes, structures and functions can be generated by local interactions between simple components. Klavins addressed this question in the field of robotics [1]. The problem is that of synchronising a population of autonomous agents and have them achieve a particular disposition in space specified as a tree. The aim of the present paper is to extend the solution given by Klavins to the case of arbitrary graphs, and to provide a formalization of the self-assembly algorithm that takes complete care of the subtler part of building a distributed consensus among agents.

The idea of the algorithm is to circulate between agents belonging to a same connected component a single copy of a mapping of their component. Whoever possesses this mapping can either pass it over to a neighbour, or decide to create a new connection, based on a successful point-to-point communication with another agent. Note that since agents are building a potentially cyclic graph, they may have to create edges to their own component. Necessary updates after a growth decision are shipped along a tree spanning the current component. Both the component and the tree are dynamically created. Interestingly, the algorithm is parameterized by the choice of a *growth scenario* specifying when an edge can be created. Thus, the solution we propose naturally supports additional constraints pertaining to which intermediate graphs are allowed during the growth of the graph.



© 2006 Kluwer Academic Publishers. Printed in the Netherlands.

The solution and the problem itself are laid down in the language of concurrency theory, and the algorithm is written in a rule-based process algebra that one could view as a simplified version of Milner’s π -calculus [5]. Although the self-assembly algorithm we present is independent of this particular choice, there is a good reason for such a formal approach. More often than not, one can go wrong in the description of such synchronisation procedures (we did quite a number of times), and the use of formal methods seems legitimate in this context, since they allow for a clear statement of correctness, and a correctness proof based on a well-established notion of equivalence known as *bisimulation*.

Our formal treatment is made relative to abstract or logical space. Including true space and explicit motorization in the agents supposes a significant extension of the usual concurrency models and as such represents an interesting challenge to formal methods. Such an extension would in particular allow a refined description of the agents behavior in the case of a group being dislocated. This is a matter to which we plan to return in a further work. For now, we provide a crude treatment of such “crashes” by introducing non deterministic alarms. The correctness of the algorithm enriched with alarms is also proved. A demo illustrating the algorithm is available on line.¹

The self-assembly question we address here was inspired by similar questions raised in the context of formal molecular biology [6, 7]. Indeed, a strong structural property that one might look for when defining a formal language for protein-protein interaction is precisely whether the formation of complexes (assemblies of proteins held together by weak bonds) can be explained in terms of only local interactions. In the context of biology, there is an additional constraint, namely that the self-assembly algorithm doesn’t build in the agents unrealistic computational prowess. With robots however, agents can be taken to be computationally strong and no such objection stays on the way of a completely satisfying result.

Results presented here were for the most part already presented in an extended abstract [8]. Section 2 presents the graph-rewriting grammar used in the following, while section 3 formulates the self-assembly problem, and presents our solution. Section 4 is new and develops a complete proof of correctness, and section 5 extends the protocol with alarms so as to escape potential deadlocks and completes the correctness proof as well.

¹ <http://www.pps.jussieu.fr/~tarissan/self>

2. Graph rewriting

In order to handle graphs and the kind of local graph rewriting our agents will perform, we introduce first a notation for graphs inspired by π -calculus, where nodes are agents, and edges are represented by name-sharing.

2.1. AGENTS AND NETWORKS


Let \mathcal{C} be a countable set of *names* ranged over by x, y, z, \dots , one defines an *agent* as a finite set $C \subset \mathcal{C}$, written $[C]$, where the set C itself is referred to as the agent *interface*. We write \mathcal{N} for the set of *agent networks* (or simply networks) defined by the following grammar:

$$G := \emptyset \mid [C] \mid G, G \mid (\nu x)G$$

where \emptyset is the empty network, G_1, G_2 stands for the juxtaposition of G_1 and G_2 , and $(\nu x)G$ stands for G where the name x has been made private to G .

Networks provide a notation for undirected graphs (without loops or multiedges): agents represent nodes, and two nodes share an edge if the corresponding agents share a name.

Here is an example:



becomes $(\nu x)(\nu y)([x], [x, y], [y])$

Our algebraic notation is redundant in that there are many distinct ways to represent the same graph. The notion of *structural congruence* below will take care of this redundancy.

The “new” operator, written in symbols ν , is a binder for names in \mathcal{C} and allows for a smooth treatment of name creation. It comes along with the usual inductive definition of *free names*:

$$\begin{aligned} \text{fn}(\emptyset) &= \emptyset \\ \text{fn}([C]) &= C \\ \text{fn}(G, G') &= \text{fn}(G) \cup \text{fn}(G') \\ \text{fn}((\nu x)G) &= \text{fn}(G) \setminus \{x\} \end{aligned}$$

An occurrence of name is said to be bound if not free. The operation of renaming bound variables is often called α -conversion.

DEFINITION 1. *Structural congruence, written \equiv , is the smallest congruence relation closed under α -conversion and such that:*

1. $(\mathcal{N}/\equiv, ', \emptyset)$ is a symmetric monoid

2. $(\nu x)(\nu y)G \equiv (\nu y)(\nu x) G$
3. $(\nu x)G \equiv G$ if $x \notin \text{fn}(G)$
4. $(\nu x)G, G' \equiv (\nu x)(G, G')$ if $x \notin \text{fn}(G')$

It is easy to see that up to structural congruence, there is a unique network representing a given isomorphism class of graphs. Based on this, we will now consider graphs as networks and don't distinguish them notationally. We also observe in passing that our notation also accommodates the description of hypergraphs.

Structural congruence allows to handle new names with set based notations, e.g., writing (νxy) in place of $(\nu x)(\nu y)$ or $(\nu y)(\nu x)$.

2.2. REACTIONS AND TRANSITION SYSTEMS

Now that we have our notation for graphs in place, we turn to the definition of a notion of graph rewriting which will be expressive enough for our needs.

DEFINITION 2. *A reaction is a pair $L, (\nu \tilde{x})R$, also written $L \rightarrow (\nu \tilde{x})R$, where $L = [L_1], \dots, [L_n]$, $R = [R_1], \dots, [R_n]$, and $\text{fn}((\nu \tilde{x})R) \subseteq \text{fn}(L)$.*

When in addition, $n \leq 2$, the reaction is said to be *local*. Local reactions express point-to-point communications and will be used to state the self-assembly problem.

Names occurring in a reaction fall naturally in three classes: the names *created* by the reaction \tilde{x} , the names *erased* by the reaction $\text{fn}(L) \setminus \text{fn}((\nu \tilde{x})R)$, and the remainder $\text{fn}(L) \cap \text{fn}((\nu \tilde{x})R)$. The condition in the definition above makes sure that any name occurring in R is either created, or already occurs in L . In the following we will suppose that $\text{fn}(L)$ and \tilde{x} are disjoint sets; this can always be realized by using α -conversion.

To fire a reaction in a network G , one looks for an instance of L in G and then replaces it with the right hand side $(\nu \tilde{x})R$. More precisely, given a set of reactions \mathfrak{R} , one defines inductively a binary relation $\rightarrow_{\mathfrak{R}}$ as follows:

$$\begin{array}{l}
 \text{(DIR)} \quad \frac{G \rightarrow_{\tau} G' \quad \tau \in \mathfrak{R}}{G \rightarrow_{\mathfrak{R}} G'} \qquad \frac{G_1 \rightarrow_{\mathfrak{R}} G_2}{G_1, G \rightarrow_{\mathfrak{R}} G_2, G} \quad \text{(GROUP)} \\
 \text{(NEW)} \quad \frac{G_1 \rightarrow_{\mathfrak{R}} G_2}{(\nu x)G_1 \rightarrow_{\mathfrak{R}} (\nu x)G_2} \quad \frac{G_1 \equiv G'_1 \quad G'_1 \rightarrow_{\mathfrak{R}} G'_2 \quad G'_2 \equiv G_2}{G_1 \rightarrow_{\mathfrak{R}} G_2} \quad \text{(STRUCT)}
 \end{array}$$

with $G \rightarrow_{\tau} G'$, if $\tau = [L_1], \dots, [L_n] \rightarrow (\nu\tilde{x})([R_1], \dots, [R_n])$, and there exists an injection i from $\text{fn}(L) \cup \tilde{x}$ to \mathcal{C} such that:

$$\begin{aligned} G &= [i(L_1)], \dots, [i(L_n)] \\ G' &= (\nu i(\tilde{x}))([i(R_1)], \dots, [i(R_n)]) \end{aligned}$$

One also writes $G \rightarrow_{\mathfrak{R}}^* G'$, whenever G' can be obtained from G by repeatedly firing reactions in \mathfrak{R} . This includes the case when no reaction is used and $G' = G$. A *transition system* is a pair (G_0, \mathfrak{R}) , where G_0 is a network, called the *initial state*, and \mathfrak{R} is a set of reactions. A transition system (G_0, \mathfrak{R}) is said to be *local* when each reaction in \mathfrak{R} is local.

2.3. SELF-ASSEMBLY

Suppose now we want our agents to self-assemble according to a given connected graph $G = (V, E)$. Write $|V|$ for the number of nodes in G , \llbracket_n for the network $\llbracket, \dots, \llbracket$ consisting of n empty agents, and $(\llbracket_n, \{r_G\})$ for the transition systems associated to G , with initial states \llbracket_n and only reaction $r_G := \langle \rangle_{|V|} \rightarrow G$.

The *self-assembly* problem for G is to find a set of *local* reactions \mathfrak{R}_G and a map θ from agents to some suitable notion of enriched agents, such that for all n , the transition system $(\theta(\llbracket_n), \mathfrak{R}_G)$ simulates—in a sense yet to be defined—the original system $(\llbracket_n, \{r_G\})$.

Two points need to be clarified here. First we need to explain how to enrich agents. An enriched agent will no longer be a mere set of names but a list of sets of names or integers. The θ function above will take care of structuring the initially empty agents according to the enriched agent format. Second, we need a definite statement about what we mean when we say that $(\theta(\llbracket_n), \mathfrak{R}_G)$ simulates $(\llbracket_n, \{r\})$, and this is where bisimulation comes in the picture.

The first point will be addressed in the course of the construction of the local transition system $(\theta(\llbracket_n), \mathfrak{R}_G)$ to which we turn now, while the second will be the object of the next section devoted to correctness.

3. The construction

Transition systems over enriched agents are similar to the ones defined before and we don't go over all the definitions of the preceding section.

We proceed to our construction in two steps. First we define the notion of a *growth scenario*, in essence a transition system describing which intermediate graphs one should seek for during the construction of the target graph G . Then we obtain the corresponding local transition system.

3.1. GROWTH SCENARIOS

Agents manipulate concrete representations of graphs and we have to be careful to distinguish these from abstract graphs. Specifically, a *concrete graph* will be taken to be a graph of the form $(\{1, \dots, n\}, E)$ with $n > 0$. The *trivial graph* $(\{1\}, \emptyset)$ will be denoted by $\mathbf{1}$. Given G a concrete graph, we write $\llbracket G \rrbracket$ for its isomorphism class, that is to say the corresponding abstract graph. Next, we define two operations on concrete graphs:

DEFINITION 3. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be concrete graphs, the join of G_1 and G_2 via $u \in V_1, v \in V_2$, written $G_1.u \oplus G_2.v$, is defined as:

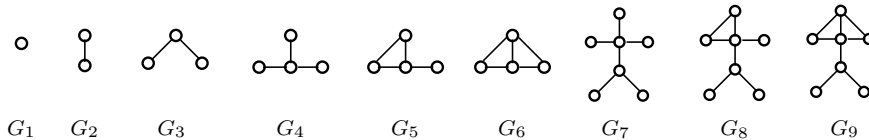
- $V = \{1, \dots, |V_1| + |V_2|\}$, and
 - $E = E_1 \cup \{\{u, v + |V_1|\}\} \cup \{\{a + |V_1|, b + |V_1|\} \mid \{a, b\} \in E_2\}$
- and the self-join of G_1 via $u, v \in V_1$, written $G_1.(u, v)$, is defined as $V = V_1$, and $E = E_1 \cup \{\{u, v\}\}$.

Note that in the binary join operation, the nodes of G_2 are shifted by $|V_1|$, and as a consequence the result is again a concrete graph. These two of join and self-join operations naturally extend to abstract graphs, and they define together a partial order, written $<$, on concrete as well as on abstract graphs.

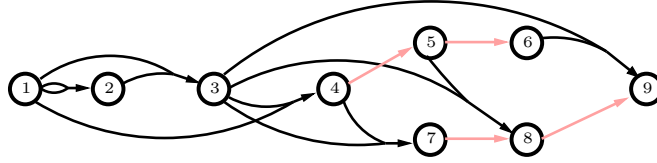
DEFINITION 4. A growth scenario \mathcal{G} is a set of abstract graphs such that for all non trivial $G \in \mathcal{G}$, either $G = G_1.u \oplus G_2.v$, for some $G_1, G_2 \in \mathcal{G}$, or $G = G_1.(u, v)$ for some $G_1 \in \mathcal{G}$.

All graphs within a scenario are connected, and conversely any connected graph is obviously contained in some scenario.

Here is an example of a growth scenario $\mathcal{G} = \{G_1, \dots, G_9\}$:



The idea is that agents wants to self-assemble to reach the target graph G_9 , and \mathcal{G} specifies all intermediate graphs they are allowed to construct in so doing. The figure below, where nodes stand for graphs in \mathcal{G} , bi-edges correspond to joins, and mono-edges to self-joins gives a visual proof that \mathcal{G} is indeed a growth scenario. Note that we do not require scenarios to be downward closed with respect to $<$, and indeed \mathcal{G} is not.



To each scenario \mathcal{G} corresponds naturally a set of reactions, denoted by $\mathfrak{R}(\mathcal{G})$, obtained by translating as reactions the joins and self-joins under which \mathcal{G} is closed. These reactions are quite specific, since they create only one name, and delete none. However they are not local. If we return to the example, the bi-edge linking G_1 and G_2 to G_3 corresponds to the following join reaction in $\mathfrak{R}(\mathcal{G})$:

$$[x], [x], [] \rightarrow (\nu y)([x], [x, y], [y])$$

3.2. THE LOCAL TRANSITION SYSTEM

With these definitions behind us, and supposing given a target graph G , and a scenario \mathcal{G} with G as only maximal graph, the self-assembly problem can now be rephrased as the problem of finding a set of local reactions that will simulate $\mathfrak{R}(\mathcal{G})$.

Let us begin with a still informal description of our algorithm and its local reactions. Each agent has in its internal state an integer called the *role* meant to describe its coordinate on the map being circulated between the agents in a same component. By allowing at any time only one agent to modify a component, we prevent concurrent modifications of the structure. This agent is chosen to be the only one holding a map of the current component, since only him needs it. So to speak, the map is itself the activity token. This makes both the internal state of an agent and the update phase simpler. Update is then reduced to the propagation of the new role played by the agents in the new enlarged component.

Going back to definition 3, we see that joins affect only the roles in one of the two connected components. Therefore, role updates are only required in one component which we take in the implementation to be the smaller one. Second, we see also that the new role is easily determined from the old one, since it is just a shift of the old one by the number of nodes, say N of the other component. Agents involved in an update phase will then simply transmit this integer N .

The case of a self-join is easier, since roles are all unchanged. Role updates are not needed, it is just the map of the active agent which is modified.

To avoid conflicts arising during update in case the graph is cyclic, we use a tree spanning the component to transmit the shift. This

spanning tree is itself a dynamic structure that grows along with the component. A new edge is added to it at each join. To reflect this structure in their states, enriched agents partition their name set in two classes, written respectively S (for span) and C (for cycles), depending on whether the corresponding edge belongs to the spanning tree or not. Take note that this tree is undirected. The direction of transmission is also dynamically determined and varies over time.

Following these informal explanations, we define the interface of an *enriched agent* as a tuple $[S, C, g, r, m]$ where S is a set of names which represents neighbours in the spanning tree, C is a set of names which represents the set of the remaining neighbours, g is a name used as a group identifier for agents in a same component, r is an integer referring to the role played by the agent in the component, and m is the agent *running mode*:

- P when the agent is in *passive* mode;
- $Act(G)$ when the agent is in *active* mode, with G a concrete graph;
- $Up(L, N)$ when the agent is in *update* mode, with L a set of names referring to the neighbours still to be updated, and N the shift.

This definition extends naturally to networks; we will write \mathcal{N}^+ for the set of *enriched networks* to distinguish it from the set \mathcal{N} of (ordinary) networks.

3.3. LOCAL REACTIONS

Given \mathcal{G} a growth scenario, we may now define our family of local reactions $\mathfrak{R}^l(\mathcal{G})$ simulating $\mathfrak{R}(\mathcal{G})$.

First come the *join reactions* (**Join**) $_{\mathcal{G}}$, with $\llbracket G_1.r_1 \oplus G_2.r_2 \rrbracket \in \mathcal{G}$:

$$\begin{array}{l} [S_1, C_1, g_1, r_1, Act(G_1)] \\ [S_2, C_2, g_2, r_2, Act(G_2)] \end{array} \longrightarrow (\nu x) \begin{array}{l} [S_1 + \{x\}, C_1, g_1, r_1, Act(G_1.r_1 \oplus G_2.r_2)] \\ [S_2 + \{x\}, C_2, g_1, r_2 + |G_1|, Up(S_2, |G_1|)] \end{array}$$

Next come the *self-join reactions* (**Self**) $_{\mathcal{G}}$, with $\llbracket G.(r_1, r_2) \rrbracket \in \mathcal{G}$:

$$\begin{array}{l} [S_1, C_1, g, r_1, Act(G)] \\ [S_2, C_2, g, r_2, P] \end{array} \longrightarrow (\nu x) \begin{array}{l} [S_1, C_1 + \{x\}, g, r_1, Act(G.(r_1, r_2))] \\ [S_2, C_2 + \{x\}, g, r_2, P] \end{array}$$

Then the *update reactions* (**Up**):

$$\begin{array}{l} [S_1, C_1, g_1, r_1, Up(L + \{x\}, N)] \\ [S_2 + \{x\}, C_2, g_2, r_2, P] \end{array} \longrightarrow \begin{array}{l} [S_1, C_1, g_1, r_1, Up(L, N)] \\ [S_2 + \{x\}, C_2, g_1, r_2 + N, Up(S_2, N)] \end{array}$$

Finally we need the *switch reactions* (**Switch**):

$$\begin{array}{l} [S_1 + \{x\}, C_1, g, r_1, Act(G)] \\ [S_2 + \{x\}, C_2, g, r_2, P] \end{array} \longrightarrow \begin{array}{l} [S_1 + \{x\}, C_1, g, r_1, P] \\ [S_2 + \{x\}, C_2, g, r_2, Act(G)] \end{array}$$

This last switch reaction, which allows the activity to circulate around in a component, can only be fired if the agents share the same group identifier. We may also note that in the update reactions, the updated agent simultaneously changes its role, and group, while entering himself in update mode. In the end-of-update reaction, the agent goes passive because his contact list is empty. It also worth noting that in the join reaction, one does not check that the group names g_1, g_2 are distinct; there is no need to do so since as we shall prove soon, there is at most one active agent per component, and therefore g_1, g_2 are distinct (in terms of process algebras this means that one doesn't use the mismatch construct).

We may now complete our definition:

DEFINITION 5. *Given a growth scenario \mathcal{G} , one defines the local transition system associated to \mathcal{G} as $(\theta(\llbracket n \rrbracket), \mathfrak{R}^l(\mathcal{G}))$, where $\mathfrak{R}^l(\mathcal{G})$ is defined above, and the initial state $\theta(\llbracket n \rrbracket)$ is given by:*

$$(g_1)[\emptyset, \emptyset, g_1, 1, Act(\mathbf{1})], \dots, (g_n)[\emptyset, \emptyset, g_n, 1, Act(\mathbf{1})]$$

A local reaction transition system can be thought of as a π -calculus process where interactions are allowed to match many names at once (whereas in π only one name is matched, that of the channel being used for sending), and information is exchanged bi-directionally (whereas in π it is uni-directional). This additional flexibility allows for a concise description of self-assembly, but it would be straightforward to translate our solution in π . Rather than doing this, we will instead import the usual π bisimulation proof method in our specific formalism.

4. Correctness

The key to proving correctness is to prove first that any active agent always has a *consistent* view on its own component. That is to say, for any P such that $\theta(\llbracket n \rrbracket) \rightarrow_{\mathfrak{R}^l(\mathcal{G})}^* P$, and any active agent in P , the image G , the group name g and the role r of the agent are always the actual ones in P .

4.1. CONSISTENT VIEW

Let $A = [S, C, g, r, m]$ be an enriched agent, we refer to $S \cup C$ as its edge names, we write $\text{span}(A) = S$ to denote names representing its span neighbourhood and $\text{group}(A) = g$ to be its group identifier.

In order to state the consistent view property, we need to handle formally the graph the enriched agents represent together with the role

they play in it. This leads to the notion of *pointed graph* (G, v) where G is a graph and v the vertex of the root.

DEFINITION 6. *Let $[S, C, g, r, m]$ be an enriched agent. We define the function σ from enriched agents to $\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{C}) \times \mathbb{N}$ by:*

$$\sigma([S, C, g, r, m]) = [S, C, n]$$

where $n = 1$ if the agent is active and 0 otherwise, the function v from $\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{C}) \times \mathbb{N}$ to agents by:

$$v([S, C, n]) = [S \cup C]$$

and the forgetting function $\rho = v \circ \sigma$ keeping only the edge names of the interface.

These definitions extend naturally to enriched networks. We will be only interested in enriched networks P such that $\sigma(P)$ denotes a *pointed spanned graph*, that is a graph with a spanning tree for each connected component (given by the set of connexions in S) and a specific node called the root and denoted by the active agent of the component. We write PSAG for the set of such pointed spanned abstract graphs and $\llbracket \sigma(P) \rrbracket$ for the PSAG corresponding to P .

The proposition below says that if an enriched network is reachable from the initial state, then the underlying network is a PSAG (condition 1), two agents sharing a common group identifier belong to the same component (condition 2), if an agent is active then its pointed graph representation (G, r) is isomorphic to its component in $\llbracket \sigma(P) \rrbracket$ (condition 3) and if an agent is in update mode then each name in the list of the agents to contact is shared with an agent in its span neighborhood (condition 4). Note that in general the converse of condition 2 does not hold since the group name may not have been propagated yet to every agents of the component.

PROPOSITION 1. (Consistency). *Suppose $\theta(\llbracket n \rrbracket) \rightarrow_{\mathfrak{R}(G)}^* P$, then:*

1. $\llbracket \sigma(P) \rrbracket \in \text{PSAG}$;
2. For all agents A_1, A_2 occurring in P , if $\text{group}(A_1) = \text{group}(A_2)$ then A_1 and A_2 belong to the same component;
3. For all agent $[S, C, g, r, \text{Act}(G)]$ occurring in P , (G, r) is isomorphic to its component in $\llbracket \sigma(P) \rrbracket$;
4. For all agent $[S, C, g, r, \text{Up}(L, N)]$ occurring in P , $L \subseteq S$, and for all x in L , there exists another agent A occurring in P such that $x \in \text{span}(A)$.

Proof. By induction on the rules of $\mathfrak{R}^l(\mathcal{G})$. The initial case is obvious since all the agents are active, disconnected, possess a different group identifier, own the initial graph $\mathbf{1}$ and play the only role possible in it. Suppose now that $\theta(\llbracket n \rrbracket) \xrightarrow{\ast_{\mathfrak{R}^l(\mathcal{G})}} P \xrightarrow{r_{\mathfrak{R}^l(\mathcal{G})}} P'$. The inductive step depends on the rule r applied:

(Join): Let $A_1 = [S_1, C_1, g_1, r_1, Act(G_1)]$ and $A_2 = [S_2, C_2, g_2, r_2, Act(G_2)]$ be the two agents involved by r . Since $\llbracket \sigma(P) \rrbracket$ is a PSAG and the two agents are active in P , it follows that they belonged to disjoint components. Then adding a name in the span neighborhood preserves the spanning structure. As only one agent remains active, we can conclude that condition 1 is preserved. Besides, this condition ensures also that for all name x in S_2 , there exists a single agent A in P' such that $x \in \text{span}(A)$. Thus, A_2 verifies condition 4 in P' . By 3, we have that (G_1, r_1) and (G_2, r_2) are in accordance with the role they play in the component. Going back to definition 3, we see that the new image $G_1.r_1 \oplus G_2.r_2$ precisely adds an edge between the nodes r_1 and r_2 and that the role of the vertex r_1 is unchanged. Thus the remaining active agent A_1 has still a consistent view $(G_1.r_1 \oplus G_2.r_2, r_1)$ of its component in P' (condition 3). Finally, condition 2 is true since the only group identifiers newly equalized are the ones of A_1 and A_2 which are now connected.

(Self): This rule adds a cycle in the graph but the spanning structure is kept unchanged as well as the activity. Since $\llbracket \sigma(P) \rrbracket$ was a PSAG (conditions 1), we deduce that this holds also for $\llbracket \sigma(P') \rrbracket$. Using a similar argument as the one in the previous case, one sees that the active agent concerned by r still verifies condition 3. Conditions 2 and 4 are not affected by the rule.

(Up): Only conditions 2 and 4 are concerned by the rule. The group identifier is changed for the receiver but condition 4 ensures that it belongs to the same component. As for the **(Join)** case, the fact that $\llbracket \sigma(P) \rrbracket$ was a PSAG ensures that this agent satisfies now condition 4. This condition also holds by induction for the sender as its set is decreased by the rule.

(End): Nothing changes except for the internal state of one agent entering a passive mode. All the conditions are satisfied by induction.

(Switch): The rule asks for the concerned agents to share a common group identifier. By condition 2 they belong then to the same component. Besides, as the shift of the role of an agent is transmitted at the same time as the group identifier, it follows that the role of the receiver corresponds indeed to its role in the component. Thus the receiver has a correct representation of the component and of the role it's playing in it (condition 3). The other conditions are not affected by the rule.

The second property one would like to check is that the enriched network only build graphs allowed by the scenario. Formally:

PROPOSITION 2. (Forward correctness). *Let P be an enriched network. If $\theta(\llbracket n \rrbracket) \rightarrow_{\mathfrak{R}^l(\mathcal{G})}^* P$, then $\llbracket n \rrbracket \rightarrow_{\mathfrak{R}(\mathcal{G})}^* \llbracket \rho(P) \rrbracket$.*

Proof. By induction on the length of the sequence leading to P . The initial case is clear since $\rho(\theta(\llbracket n \rrbracket)) = \llbracket n \rrbracket$. Suppose now that $\theta(\llbracket n \rrbracket) \rightarrow_{\mathfrak{R}^l(\mathcal{G})}^* P \xrightarrow{r}_{\mathfrak{R}^l(\mathcal{G})} P'$. We reason by case on r .

If a connection is created then the agents involved by the rule had a consistent view in P of the component they belonged to (by condition 3 of proposition 1). The conditions required for the rule to be applied, namely $\llbracket G_1.r_1 \oplus G_2.r_2 \rrbracket \in \mathcal{G}$ for (**Join**) and $\llbracket G.(r_1, r_2) \rrbracket \in \mathcal{G}$ for (**Self**), ensure explicitly that a binary join or a unary join can be triggered in $\llbracket \rho(P) \rrbracket$ leading to $\llbracket \rho(P') \rrbracket$.

If no creation is created, then the rule don't modify the structure of the components and $\llbracket \rho(P') \rrbracket = \llbracket \rho(P) \rrbracket$.

4.2. BISIMULATION

Now we are ready to explain in which sense our local transition system, $\mathfrak{R}^l(\mathcal{G})$, behaves as the global one, $\mathfrak{R}(\mathcal{G})$. This asks first for defining a relation between the states of the two systems.

For any reaction r in $\mathfrak{R}(\mathcal{G})$, write $Inter(r)$ for the set of joins or self-joins in $R^l(G)$ corresponding to r .

DEFINITION 7. (Relation $\approx_{\mathcal{G}}$). *The rules*

$$\begin{array}{l}
 \text{(Init)} \quad \frac{}{\llbracket n \rrbracket \approx_{\mathcal{G}} \theta(\llbracket n \rrbracket)} \\
 \text{(Inter)} \quad \frac{S \approx_{\mathcal{G}} T \quad S \xrightarrow{s}_{\mathfrak{R}(\mathcal{G})} S' \quad T \xrightarrow{t}_{\mathfrak{R}^l(\mathcal{G})} T' \quad t \in Inter(s)}{S' \approx_{\mathcal{G}} T'} \\
 \text{(Invis)} \quad \frac{S \approx_{\mathcal{G}} T \quad T \xrightarrow{t}_{\mathfrak{R}^l(\mathcal{G})} T' \quad t \in \{\text{Up, End, Switch}\}}{S \approx_{\mathcal{G}} T'}
 \end{array}$$

inductively define a binary relation $\approx_{\mathcal{G}}$ on $\mathcal{N} \times \mathcal{N}^+$.

Notice that the relation preserves the correspondence between networks exhibiting the same underlying graphs.

PROPOSITION 3. *If $P \approx_{\mathcal{G}} P'$, then $\llbracket P \rrbracket \simeq \llbracket \rho(P') \rrbracket$.*

Proof. Straightforward induction on $\approx_{\mathcal{G}}$.

Then, we use the notion of bisimulation to compare the behaviors of the two systems. If $\mathcal{S} = (P_0, \mathfrak{R})$ is a transition system, we write $\mathbf{re}(\mathcal{S}) = \{P \mid P_0 \xrightarrow{*}_{\mathfrak{R}} P\}$ for the set of its reachable states.

DEFINITION 8. *Two transition systems $\mathcal{S} = (P_0, \mathfrak{R})$ and $\mathcal{S}' = (P'_0, \mathfrak{R}')$ are bisimilar if there exists a binary relation \approx over $\mathbf{re}(\mathcal{S}) \times \mathbf{re}(\mathcal{S}')$ such that $P_0 \approx P'_0$ and whenever $P \approx P'$:*

- if $P \xrightarrow{\mathfrak{R}} Q$, then there exists Q' such that $P' \xrightarrow{*}_{\mathfrak{R}'} Q'$ and $Q \approx Q'$;
- if $P' \xrightarrow{\mathfrak{R}'} Q'$, then there exists Q such that $P \xrightarrow{*}_{\mathfrak{R}} Q$ and $Q \approx Q'$.

The rest of this section is devoted to proving that $\approx_{\mathcal{G}}$ is bisimulation. Note that the obtained property is slightly more general than the one we wanted, since it does not assume that there is a unique maximal graph in the scenario \mathcal{G} .

The first condition is ensured by the **(Init)** rule. The only difficulty which remains here is that the enriched network may not be able to simulate immediately an interaction rule because updates may not have been transmitted to the concerned agents yet. To handle this case, we define for every enriched network P the *cleaned state* of P (written $\mathbf{cl}(P)$) as the network obtained by repeated applications of **(Up)** and **(End)** rules.

DEFINITION 9. (Cleaning the system). *Let P be an enriched network. We define $\mathbf{cl}(P)$ as the network verifying:*

- $P \xrightarrow{*} \mathbf{cl}(P)$ using only rules **(Up)** and **(End)**;
- $\forall r \in \{\mathbf{Up}, \mathbf{End}\} \forall P' \mathbf{cl}(P) \xrightarrow{r} P'$.

Note that the system is locally confluent since updating an agent doesn't prevent the application of another updating rule. Moreover, it follows from conditions 1 and 4 of proposition 1 that the structure used to transmit the update is a tree, ensuring in particular that there is no loop. Since the set of agents to contact is decreasing while using the rules, the process terminates. We deduce that the system is confluent and that \mathbf{cl} defines a unique $\mathbf{cl}(P)$.

Of course, as we already noticed these steps of computation don't change the underlying graphs neither the images carried by the agents. Clearly by rule **(Invis)**:

LEMMA 1. *If $P \approx_{\mathcal{G}} Q$, then $P \approx_{\mathcal{G}} \text{cl}(Q)$.*

Finally, we present and prove our main result :

THEOREM 1. (Correctness). *For all n and all growth scenarios \mathcal{G} , the global and local transition systems, $(\llbracket_n, \mathfrak{R}(\mathcal{G})\rrbracket)$ and $(\theta(\llbracket_n), \mathfrak{R}^l(\mathcal{G}))$, are bisimilar.*

Proof. We prove that $\approx_{\mathcal{G}}$ is a bisimulation. The first condition is directly satisfied by the rule (Init).

Suppose now that $P \approx_{\mathcal{G}} Q$ and that $P \xrightarrow{r}_{\mathfrak{R}(\mathcal{G})} P'$ for some r and P' . By lemma 1 we have that $P \approx_{\mathcal{G}} \text{cl}(Q)$. Depending on whether the active agents in $\text{cl}(Q)$ are the ones concerned by r or not, the system $\text{cl}(Q)$ may have to use the (Switch) rule to pass the token to the concerned agents, leading the system to Q' . By (Invis), $P \approx_{\mathcal{G}} Q'$ and then by proposition 3 it follows that P and Q' have the exact same graphs. Now there is a rule $r' \in \text{Inter}(r)$ which can be applied to Q' leading to Q'' and $P' \approx_{\mathcal{G}} Q''$ by application of the rule (Inter).

Conversely, suppose that $P \approx_{\mathcal{G}} Q$ and $Q \xrightarrow{r}_{\mathfrak{R}^l(\mathcal{G})} Q'$ for some r and Q' . We show that there exists P' such that $P \xrightarrow{*}_{\mathfrak{R}(\mathcal{G})} P'$ and $P' \approx_{\mathcal{G}} Q'$ by reasoning on r :

- $r \in \text{Inter}(r')$: This is a consequence of proposition 3. Since they exhibit the same graphs, r' can be triggered in P leading the system to P' . Then we prove that $P' \approx_{\mathcal{G}} Q'$ by application of the rule (Inter).
- $r \in \{\text{Up}, \text{End}, \text{Switch}\}$: We take $P' = P$ and we conclude with an application of the rule (Invis).

5. Escaping deadlocks

The local transition systems defined above are monotonic in the sense that edges can only be added. Different components representing partially grown target graphs could compete and be deprived of resources.

Klavins suggests a simple timeout method to grow, starting with a given population of agents, the maximum possible number of copies of the target graph. We incorporate now an abstract version of this deadlock escape mechanism in our algorithm.

First we extend our notion of growth scenario by allowing each connected component to dislocate in one step, defining thus an extended global transition system $\mathfrak{R}^e(\mathcal{G})$. Second, we introduce a new

alarm mode, written Al . And third, we add the accompanying reactions, starting with the *breaking-loose reactions* (**Break**):

$$[S, C, g, r, Act(G)] \longrightarrow [S, C, g, r, Al]$$

and the *alarm propagation reactions* (**Prop**):

$$\begin{array}{l} [S_1 + \{x\}, C_1, g_1, r_1, Al], \\ [S_2 + \{x\}, C_2, g_2, r_2, -] \end{array} \longrightarrow \begin{array}{l} [S_1, C_1, g_1, r_1, Al], \\ [S_2, C_1, g_2, r_2, Al] \end{array}$$

$$\begin{array}{l} [S_1, C_1 + \{x\}, g_1, r_1, Al], \\ [S_2, C_2 + \{x\}, g_2, r_2, -] \end{array} \longrightarrow \begin{array}{l} [S_1, C_1, g_1, r_1, Al], \\ [S_2, C_1, g_2, r_2, Al] \end{array}$$

and finally the *alarm-end reactions* (**EndAl**):

$$[\emptyset, \emptyset, g, c, Al] \longrightarrow (\nu g)[\emptyset, \emptyset, g, 1, Act(\mathbf{1})]$$

Note that one has two reactions to propagate the alarm depending on whether the alarm is shipped along the spanning tree or not. There are no longer any conditions on g_2 and r_2 , since consistency is lost during dislocation. An agent goes active again only when it has broken all connections and spread the alarm to all its neighbours. Thus, active agents still view correctly their components.

The correctness of this extended algorithm is based on the one given section 4. Specifically, if we call $\mathfrak{R}^{al}(\mathcal{G})$ the system of rules based on $\mathfrak{R}^l(\mathcal{G})$ with the addition of the three rules above, one would like to show that $(\llbracket_n, \mathfrak{R}^e(\mathcal{G})\rrbracket)$ and $(\theta(\llbracket_n), \mathfrak{R}^{al}(\mathcal{G}))$ are bisimilar.

We won't rehash here all the proofs we made earlier but only give the minimal definitions and reformulations of the main properties in order to handle the new situation. The main point will be to keep the same correspondence as before between the two networks as long as no dislocations are engaged in the enriched one. As soon as an alarm propagates, we consider the component as lost.

DEFINITION 10. (Stabilizing the system). *Let P be an enriched network. We define $st(P)$ as the network verifying:*

$$-P \rightarrow^* st(P) \text{ using only rules (Prop) and (EndAl);}$$

$$-\forall r \in \{\mathbf{Prop}, \mathbf{EndAl}\} \forall P' st(P) \xrightarrow{r} P'.$$

Since a name is erased from the edge set as soon as it is used to propagate the alarm, the set of agents to contact decreases and that the agent can't be forced to enter the alarm mode again when it has finished. As for definition 9, this operation on enriched network is confluent. Besides, since an alarm is triggered only by active agents,

if an agent is active, then no alarm is running in its component and the agent still has a consistent view of it. Thus for any enriched network P , $\text{st}(P)$ satisfies the conditions of proposition 1.

It remains now to exhibit as before a bisimulation between the two networks able extend the theorem 1 to include the case of dislocation. This new relation, namely $\approx_{\mathcal{G}}^{\text{al}}$, will be built on $\approx_{\mathcal{G}}$ by adding a rule (**Restart**) which states that in case of dislocation, one forgets all the connections of the component and considers the agents as disconnected and active again.

DEFINITION 11. *The rules*

$$\overline{\llbracket n \approx_{\mathcal{G}}^{\text{al}} \theta(\llbracket n) \rrbracket}$$

$$\frac{S \approx_{\mathcal{G}}^{\text{al}} T \quad S \xrightarrow{s}_{\mathfrak{R}(\mathcal{G})} S' \quad T \xrightarrow{t}_{\mathfrak{R}^{\text{al}}(\mathcal{G})} T' \quad t \in \text{Inter}(s)}{S' \approx_{\mathcal{G}}^{\text{al}} T'}$$

$$\frac{S \approx_{\mathcal{G}}^{\text{al}} T \quad T \xrightarrow{t}_{\mathfrak{R}^{\text{al}}(\mathcal{G})} T' \quad t \in \{\text{Up} \cup \text{End} \cup \text{Switch} \cup \text{Prop} \cup \text{EndAl}\}}{S \approx_{\mathcal{G}}^{\text{al}} T'}$$

$$\frac{S \approx_{\mathcal{G}}^{\text{al}} T \quad T \xrightarrow{t}_{\mathfrak{R}^{\text{al}}(\mathcal{G})} T' \quad S' = \text{st}(T') \quad t = \text{Break}}{S' \approx_{\mathcal{G}}^{\text{al}} T'}$$

inductively define a binary relation $\approx_{\mathcal{G}}^{\text{al}}$ on $\mathcal{N} \times \mathcal{N}^+$.

It is now easy to see that this relation is actually a bisimulation. It is enough to notice that as soon as a connected component has begun to dislocate, it has no choice but keeping breaking all the connections before becoming active again.

THEOREM 2. (Correctness). *For all n and all growth scenarios \mathcal{G} , the global and local transition systems, $(\llbracket n, \mathfrak{R}^e(\mathcal{G}) \rrbracket)$ and $(\theta(\llbracket n), \mathfrak{R}^{\text{al}}(\mathcal{G}))$, are bisimilar.*

As a corollary we obtained a complete solution of the self-assembly problem for G .

COROLLARY 1. *For all n , all graphs G and all growth scenarios \mathcal{G} having G as top element, the original and local transition systems, $(\llbracket n, \{r_G\} \rrbracket)$ and $(\theta(\llbracket n), \mathfrak{R}^{\text{al}}(\mathcal{G}))$, are bisimilar.*

Proof. One can always simulate the step $\llbracket_{|V|} \rightarrow G$ by dislocating as many components not isomorphic to G as needed to use the agents according to a chosen path in \mathcal{G} leading to G .

6. Conclusion

We have presented an abstract self assembly protocol by which a population of autonomous agents can grow an arbitrary network of connections in a distributed way. The target network is built incrementally. In any given connected component, a map of the current component circulates between the agents allowing them to make decisions concerning whether an edge should be added and where. One specificity of our approach is to cast the problem in the language of concurrent processes, actually in a simplified version of π -calculus, and be able to subsequently give a precise statement of the correctness of the proposed protocol. An implementation of the protocol extended with an alarm propagation mechanism avoiding deadlocks is available online.² A decidedly interesting extension would be to incorporate true space, if only because no ordinary process algebra models true space.

One may wonder whether there are means to escape deadlock other than the method used here. The problem is related to programming the backward behavior of the agents. In particular it requires synchronizations erasing connections while preserving a consistent view of the component the agents belong to. A general backtracking strategy has been investigated in [9] where a subtle backtracking mechanism is proposed for the language of description itself. This allows to let the question of deadlocks aside when programming and gives a correctness of the general mechanism which doesn't need to be reinvented for each application.

The self-assembly protocol was inspired by self assembly questions in biological systems. Although we assumed computationally strong agents, we took care to keep their internals and interaction capacities, as embodied by the local reactions, to a minimum. It is agreed that they are still way stronger than anything that could be implemented today in the combinatorics of molecular biology. Nevertheless exploring such self-assembly procedures could help in building a working engineering intuition of biological self assembly.

² <http://www.pps.jussieu.fr/~tarissan/self>

References

1. Eric Klavins. Automatic synthesis of controllers for assembly and formation forming. In *Proceedings of the International Conference on Robotics and Automation*, 2002.
2. Eric Drexler and Richard Smalley. Controversy about molecular assemblers. Available at www.foresight.org/NanoRev/Letter.html, 2003.
3. Radhika Nagpal. Programmable self-assembly using biologically-inspired multi-agent control. In *Autonomous Agents and Multiagent Systems Conference (AAMAS)*, July 2002.
4. Jeff Hasty, David McMillen, and James J. Collins. Engineered gene circuits. *Nature*, 420:224–230, November 2002.
5. Robin Milner. *Communicating and mobile systems: the π -calculus*. Cambridge University Press, Cambridge, 1999.
6. Vincent Danos and Cosimo Laneve. Core formal molecular biology. In *Proceedings of the 12th European Symposium on Programming (ESOP'03, Warsaw, Poland)*, volume 2618 of *LNCS*, pages 302–318. Springer, April 2003.
7. Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, September 2004.
8. Vincent Danos and Fabien Tarissan. Self assembling graphs. In *Mechanisms, Symbols, and Models Underlying Cognition*, volume 3561 of *LNCS*, pages 498–507. Springer, 2005.
9. Vincent Danos, Jean Krivine, and Fabien Tarissan. Self assembling trees. In *Proceedings of the 2005 Artificial Evolution Conference*. Springer. to appear.