# Towards distributed bigraphical reactive systems
## Distributed computing of bigraphical embeddings*

Alessio Mansutti Marco Peressotti Marino Miculan

Laboratory of Models and Applications of Distributed Systems,
Department of Mathematics and Computer Science, University of Udine, Italy
alessio.mansutti@gmail.com, {marco.peressotti, marino.miculan}@uniud.it

**Abstract.** The bigraph embedding problem is crucial for many results and tools about bigraphs and bigraphical reactive systems (BRS). There are algorithms for computing bigraphical embedding but these are designed to be run locally and assume a complete view of the guest and host bigraphs, putting large bigraphs and BRS out of their reach. To overcome these limitations we present a *decentralized algorithm* for computing bigraph embeddings that allows us to distribute both state and computation over several concurrent processes. Among various applications, this algorithm offers the basis for distributed BRS simulations where non-interfering reactions are carried out concurrently.

## 1 Introduction

*Bigraphical Reactive Systems* (BRSs) [10,15] are a flexible and expressive meta-model for ubiquitous computation. In the last decade, BRSs have been successfully applied to the formalization of a wide range of domain-specific calculi and models, from traditional programming languages to process calculi for concurrency and mobility, from business processes to systems biology; a non exhaustive list is [1,3,4,6,12,13]. Recently, BRSs have found a promising applications in *structure-aware agent-based computing:* the knowledge about the (physical) world where the agents operate (e.g., drones, robots, etc.) can be conveniently represented by means of BRSs [16,20]. BRSs are appealing also because they provide a range of general results and tools, which can be readily instantiated with the specific model under scrutiny: simulation tools, systematic construction of compositional bisimulations [10], graphical editors [7], general model checkers [18], modular composition [17], stochastic extensions [11], etc.

This expressive power stems from the rich structure of *bigraphs*, which form the states of a bigraphic reactive system. A bigraph is a compositional data structure describing at once both the locations and the logical connections of (possibly nested) components of a system. To this end, bigraphs combine two independent graphical structures over the same set of *nodes*: a hierarchy of *places*, and a hypergraph of *links*. Intuitively, places can be used for representing physical positions of agents, while links represent logical connections between agents. A simple example is shown in Figure 1.

* Work partially supported by MIUR PRIN project 2010LHT4KM, *CINA*.

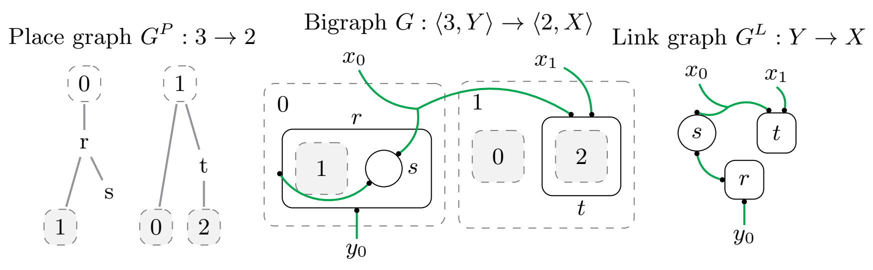
Place graph $G^P : 3 \to 2$
Bigraph $G : \langle 3, Y\rangle \to \langle 2, X\rangle$
Link graph $G^L : Y \to X$
0
1
r
s
t
1
0
2
$x_0$
$x_1$
$y_0$
$r$
$s$
$t$

**Fig. 1.** Forming a bigraph from a place graph and a link graph.

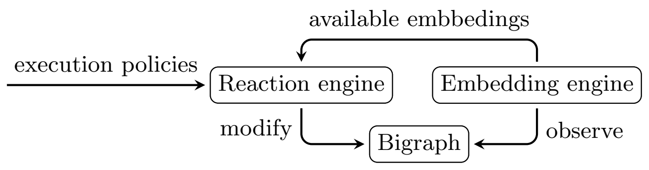

**Fig. 2.** An abstract bigraphical machine.

Like graph rewriting [19], the behaviour of a BRS is defined by a set of *(parametric) reaction rules*, which can modify a bigraph by replacing a *redex* with a *reactum*, possibly changing agents' positions and connections.

Bigraphical reactive systems can be run (or simulated) by the abstract machine depicted in Figure 2 (or variants of it). This machine is composed by two main modules: the *embedding engine* and the *reaction engine*. The former is responsible of keeping track of every occurrence of the redexes into the machine state. The latter is responsible of carrying out the reactions, in two steps: (a) choosing an occurrence of a redex among those provided by the embedding engine and (b) updating the machine state by performing the chosen rewrite operation. The selection of the reaction is driven by user-provided execution policies.

Therefore, computing bigraph embeddings is a central issue in any implementation of a BRS abstract machine. The problem is known to be NP-complete [2], and some algorithms (or reductions) can be found in the literature [8, 14, 21]. However, existing algorithms assume a complete view of both the guest and the host bigraphs. This hinders the scalability of BRS execution tools, especially on devices with low resources (like embedded ones). Moreover, in a truly distributed setting (like in multi-agent systems [12]) the bigraph is scattered among many machines; gathering it to a single "knowledge manager" in order to calculate embeddings and apply the rewriting rules, would be impractical.

In this paper, we aim to overcome these problems, by introducing an algorithm for computing bigraphical embeddings in distributed settings where bigraphs are spread across several cooperating processes. This decentralized algorithm does not impose a complete view of the host bigraph, but retains the fundamental property of (eventually) computing every possible embedding for

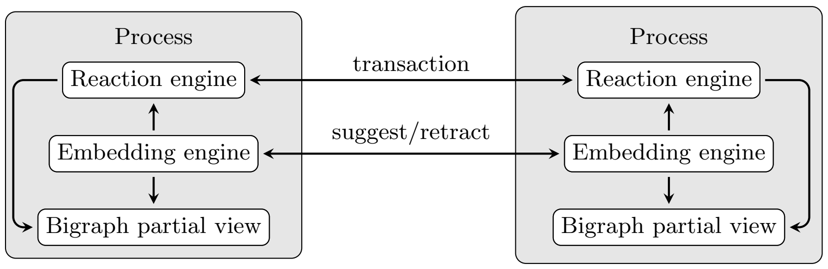

**Fig. 3.** Distributed bigraphical machine.

the given host. Thanks to the decentralized nature of the algorithm, this solution can scale to bigraphs that cannot fit into the memory of a single process, hence too large to be handled by existing implementations. Moreover, the algorithm is parallelized: several (non-interfering) reductions can be identified and applied at once. In this paper we consider distributed hosts only since guests are usually redexes of parametric reaction rules and hence small enough to be handled even in presence of scarce computational resources.

Thanks to this result we are able to define a decentralized variation of the abstract bigraphical machine illustrated above. The architecture of this new *distributed bigraphical machine* is sketched in Figure 3. Both computation and states are distributed over a family of processes. Each process has only a partial view of the global state and negotiates updates to its piece of the global bigraph with its "neighbouring processes". In order to simplify the exposition we assume reliable asynchronous point-to-point communication between reliable processes. These are mild assumptions for a distributed system and can be easily achieved e.g. over unreliable channels.

*Synopsis* In Section 2 we briefly recall the notion of bigraphs and bigraphical reactive systems. In Section 3 we recall the notion of bigraph embedding, introduce the notion of *partial embedding* and study their ordering (which are at the base of our algorithm). In Section 4 and Section 5 we describe the distributed bigrapical machine and its components; especially the distributed algorithm for solving the embedding problem at the core of this paper. Conclusions and final remarks are discussed in Section 6.

## 2 Bigraphical reactive systems

In this section we briefly recall the notion of Bigraphical Reactive Systems (BRS) referring the interested reader to [15]. The key point of BRSs is that "the model should consist in some sort of reconfigurable space". Agents may interact in this space, even if they are spatially separated. This means that two agents may be adjacent in two ways: they may be at the same *place*, or they may be connected by a *link*. This leads to the definition of *bigraphs* as a data structure

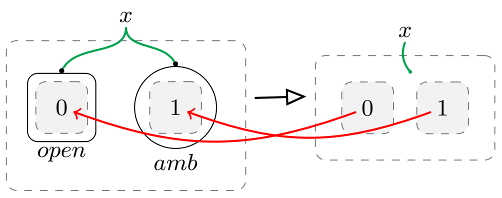

**Fig. 4.** Open reaction rule of the Ambient Calculus.

for representing the state of the system. A bigraph can be seen as an enriched hyper-graph combining two independent graphical structures over the same set of *nodes*: a hierarchy of *places*, and a hyper-graph of *links*.

**Definition 1 (Bigraph [15, Def. 2.3]).** *Let $\Sigma$ be a bigraphical signature (i.e. a set of types, called* controls*, denoting a finite arity). A* bigraph *$G$ over $\Sigma$ is an object $(V_G, E_G, \mathrm{ctrl}_G, \mathrm{prnt}_G, \mathrm{link}_G) : \langle m_G, X_G\rangle \to \langle n_G, Y_G\rangle$ composed of two substructures (cf. Figure 1): a* place graph *$G^P = (V_G, \mathrm{ctrl}_G, \mathrm{prnt}_G) : m_G \to n_G$ and a* link graph *$G^L = (V_G, E_G, \mathrm{ctrl}_G, \mathrm{link}_G) : X_G \to Y_G$. The set $V_G$ is a finite set of nodes and to each of them is assigned a control in $\Sigma$ by the* control map *$\mathrm{ctrl}_G : V_G \to \Sigma$. The set $E_G$ is a finite set of names called* edges*.*

*These structures present an inner interface (composed by $m_G$ and $X_G$) and an outer one ($n_G$, $Y_G$) along which can be composed with other of their kind as long as they do not share any node or edge. In particular, $X_G$ and $Y_G$ are finite sets of names and $m_G$ and $n_G$ are finite ordinals.*

*On the side of $G^P$, nodes, sites and roots are organized in a forest described by the* parent map *$\mathrm{prnt}_G : V_G \uplus m_G \to V_G \uplus n_G$ s.t. sites are leaves and roots are $n_G$.*

*On the side of $G^L$, nodes, edges and names of the inner and outer interface forms a hyper-graph described by the* link map *$\mathrm{link}_G : P_G \uplus X_G \to E_G \uplus Y_G$ which is a function from $X_G$ and ports $P_G$ (i.e. elements of the finite ordinal associated to each node by its control) to edges $E_G$ and names in $Y_G$.*

The dynamic behaviour of a system is described in terms of *reactions* of the form $a \rightarrowtriangle a'$ where $a, a'$ are agents, i.e. bigraphs with inner interface $\langle 0, \emptyset\rangle$. Reactions are defined by means of graph rewrite rules, which are pairs of bigraphs $(R_L, R_R)$ equipped with a function $\eta$ from the sites of $R_R$ to those of $R_L$ called *instantiation rule*. A bigraphical encoding for the open reaction rule of the Ambient Calculus is shown in Figure 4 where redex and reactum are the bigraph on the left and the one on the right respectively and the instantiation rule is drawn in red. A rule fires when its redex can be embedded into the agent; then, the matched part is replaced by the reactum and the parameters (i.e. the substructures determined by the redex sites) are instantiated accordingly with $\eta$.

## 3 Partial bigraph embeddings

The following definitions are mainly taken from [9], with minor modification to simplify the presentation of the distributed embedding algorithm (cf. Section 5).

As usual, we will exploit the orthogonality of the link and place graphs, by defining *link and place graph embeddings* separately and then combine them to extend the notion to bigraphs. We then introduce *partial bigraph embeddings*, define an ordering on them and study the atomic $CPO_\perp$ structure presented by the set of partial embeddings for any given pair of guest and host bigraphs. This structure is fundamental for the algorithm we present in Section 5.

*Link graph* Intuitively an embedding of link graphs is a structure preserving map from one link graph (the *guest*) to another (the *host*). As one would expect from a graph embedding, this map contains a pair of injections: one for the nodes and one for the edges (i.e., a support translation). The remaining of the embedding map specifies how names of the inner and outer interfaces should be mapped into the host link graph. Outer names can be mapped to any link; here injectivity is not required since a context can alias outer names. Dually, inner names can mapped to hyper-edges linking sets of points in the host link graph and such that every point is contained in at most one of these sets.

**Definition 2 (Link graph embedding [9, Def 7.5.1]).** *Let $G : X_G \to Y_G$ and $H : X_H \to Y_H$ be two concrete link graphs. A* link graph embedding $\phi : G \hookrightarrow H$ *is a map $\phi \triangleq \phi^{\mathsf{v}} \uplus \phi^{\mathsf{e}} \uplus \phi^{\mathsf{i}} \uplus \phi^{\mathsf{o}}$ (assigning nodes, edges, inner and outer names respectively) subject to the following conditions:*

**(LGE-1)** $\phi^{\mathsf{v}} : V_G \rightarrowtail V_H$ *and* $\phi^{\mathsf{e}} : E_G \rightarrowtail E_H$ *are injective;*
**(LGE-2)** $\phi^{\mathsf{i}} : X_G \rightarrowtail \wp(X_H \uplus P_H)$ *is fully injective:* $\forall x \neq x' : \phi^{\mathsf{i}}(x) \cap \phi^{\mathsf{i}}(x') = \emptyset$*;*
**(LGE-3)** $\phi^{\mathsf{o}} : Y_G \to E_H \uplus Y_H$ *in an arbitrary partial map;*
**(LGE-4)** $\mathrm{img}(\phi^{\mathsf{e}}) \cap \mathrm{img}(\phi^{\mathsf{o}}) = \emptyset$ *and* $\mathrm{img}(\phi^{\mathsf{i}}) \cap \mathrm{img}(\phi^{\mathsf{port}}) = \emptyset$*;*
**(LGE-5)** $\phi^{\mathsf{p}} \circ \mathrm{link}_G^{-1}\big|_{E_G} = \mathrm{link}_H^{-1} \circ \phi^{\mathsf{e}}$*;*
**(LGE-6)** $\mathrm{ctrl}_G = \mathrm{ctrl}_H \circ \phi^{\mathsf{v}}$*;*
**(LGE-7)** $\forall p \in X_G \uplus P_G : \forall p' \in (\phi^{\mathsf{p}})(p) : (\phi^{\mathsf{h}} \circ \mathrm{link}_G)(p) = \mathrm{link}_h(p')$

*where* $\phi^{\mathsf{p}} \triangleq \phi^{\mathsf{i}} \uplus \phi^{\mathsf{port}}$, $\phi^{\mathsf{h}} \triangleq \phi^{\mathsf{e}} \uplus \phi^{\mathsf{o}}$ *and* $\phi^{\mathsf{port}} : P_G \rightarrowtail P_H$ *is* $\phi^{\mathsf{port}}(v, i) \triangleq (\phi^{\mathsf{v}}(v), i))$.

The first three conditions are on the single sub-maps of the embedding. Condition (LGE-4) ensures that no components (except for outer names) are identified; condition (LGE-5) imposes that points connected by the image of an edge are all covered. Finally, conditions (LGE-6) and (LGE-7) ensure that the guest structure is preserved i.e. node controls and point linkings are preserved.

*Place graph* Like link graph embeddings, place graph embeddings are just a structure preserving injective map from nodes along with suitable maps for the inner and outer interfaces. In particular, a site is mapped to the set of sites and nodes that are "put under it" and a root is mapped to the host root or node that is "put over it" splitting the host place graphs in three parts: the guest image, the context and the parameter (which are above and below the guest image).

**Definition 3 (Place graph embedding [9, Def 7.5.4]).** *Let $G : n_G \to m_G$ and $H : n_H \to m_H$ be two concrete place graphs. A* place graph embedding $\phi : G \hookrightarrow H$ *is a map $\phi \triangleq \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}}$ (assigning nodes, sites and regions respectively) subject to the following conditions:*

**(PGE-1)** $\phi^{\mathsf{v}} : V_G \rightarrowtail V_H$ *is injective;*
**(PGE-2)** $\phi^{\mathsf{s}} : n_G \rightarrowtail \wp(n_H \uplus V_H)$ *is fully injective;*
**(PGE-3)** $\phi^{\mathsf{r}} : m_G \to V_H \uplus m_H$ *in an arbitrary map;*
**(PGE-4)** $\mathrm{img}(\phi^{\mathsf{v}}) \cap \mathrm{img}(\phi^{\mathsf{r}}) = \emptyset$ *and* $\mathrm{img}(\phi^{\mathsf{s}}) \cap \mathrm{img}(\phi^{\mathsf{v}}) = \emptyset$*;*
**(PGE-5)** $\forall r \in m_G : \forall s \in n_G : \mathrm{prnt}^*_H \phi^{\mathsf{r}}(r) \cap \phi^{\mathsf{s}}(s) = \emptyset$*;*
**(PGE-6)** $\phi^{\mathsf{c}} \circ \mathrm{prnt}_G^{-1}\big|_{V_G} = \mathrm{prnt}_H^{-1} \circ \phi^{\mathsf{v}}$*;*
**(PGE-7)** $\mathrm{ctrl}_G = \mathrm{ctrl}_H \circ \phi^{\mathsf{v}}$*;*
**(PGE-8)** $\forall c \in n_G \uplus V_G : \forall c' \in \phi^{\mathsf{c}}(c) : (\phi^{\mathsf{f}} \circ \mathrm{prnt}_G)(c) = \mathrm{prnt}_H(c')$*;*

*where* $\phi^{\mathsf{f}} \triangleq \phi^{\mathsf{v}} \uplus \phi^{\mathsf{r}}$ *and* $\phi^{\mathsf{c}} \triangleq \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}}$*.*

Conditions in the above definition follows the structure of Definition 2, the main notable difference is (PGE-5) which states that the image of a root can not be the descendant of the image of another. Conditions (PGE-1), (PGE-2) and (PGE-3) are on the three sub-maps composing the embedding; conditions (PGE-4) and (PGE-5) ensure that no components are identified; (PGE-6) imposes surjectivity on children and the last two conditions require the guest structure to be preserved by the embedding map.

*Bigraph* Finally, bigraph embeddings can now be defined as maps being composed by an embedding for the link graph with one for the place graph consistently with the interplay of these two substructures. In particular, the interplay is captured by a single additional condition ensuring that points in the image of an inner names reside in the parameter defined by the place graph embedding (i.e. are inner names or ports of some node under a site image).

**Definition 4 (Bigraph embedding [9, Def 7.5.14]).** *Let* $G : \langle n_G, X_G\rangle \to \langle m_G, Y_G\rangle$ *and* $H : \langle n_H, X_H\rangle \to \langle m_H, Y_H\rangle$ *be two concrete bigraphs. A* bigraph embedding $\phi : G \hookrightarrow H$ *is a map given by a place graph embedding* $\phi^{\mathsf{P}} : G^P \hookrightarrow H^P$ *and a link graph embedding* $\phi^{\mathsf{L}} : G^L \hookrightarrow H^L$ *subject to the consistency condition:*

**(BGE-1)** $\mathrm{img}(\phi^{\mathsf{i}}) \subseteq X_H \uplus \{(v,i) \in P_H \mid \exists s \in n_G : k \in \mathbb{N} : \mathrm{prnt}^k_H(v) \in \phi^{\mathsf{s}}(s)\}$.

*Partial bigraph embeddings* In the following we relax the above definition to allow partiality and formally represent intermediate steps of the algorithm we present in Section 5. Basically, a partial bigraph embedding is a partial map subject to the same conditions of a total embedding up-to partiality.

**Definition 5 (Partial bigraph embedding).** *Let* $G : \langle n_G, X_G\rangle \to \langle m_G, Y_G\rangle$ *and* $H : \langle n_H, X_H\rangle \to \langle m_H, Y_H\rangle$ *be two concrete bigraphs. A* partial bigraph embedding $\phi : G \hookrightharpoonup H$ *is a partial map subject, where defined, to the same conditions of Definition 4.*

Partial embeddings represent partial or intermediate steps towards a total embedding. This is reflected by the obvious ordering given by the point-wise lifting of the anti-chain order to partial maps. In particular, given two partial embeddings $\phi, \psi : G \hookrightharpoonup H$ we say that:

$$\phi \sqsubseteq \psi \overset{\triangle}{\iff} \forall x \in \mathrm{dom}(\phi)\, \phi(x) \neq \bot \implies \psi(x) = \phi(x). \tag{1}$$

This definition extends, for any given pair of concrete bigraphs $G$ and $H$, to a partial order over the set of partial bigraph embeddings of $G$ into $H$. It is easy to check that the entirely undefined embedding $\varnothing$ is the bottom of this structure and that meets are always defined:

$$\phi \sqcap \psi \triangleq \lambda x. \begin{cases} \phi(x) & \text{if } \phi(x) = \psi(x) \\ \bot & \text{otherwise} \end{cases}$$

Likewise, joins, where they exist, are defined as follows:

$$\phi \sqcup \psi \triangleq \lambda x. \begin{cases} \phi(x) & \text{if } \phi(x) \neq \bot \\ \psi(x) & \text{if } \psi(x) \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

Clearly $\phi$ and $\psi$ have to coincide where are both defined and their join $\phi \sqcup \psi$ is defined iff it meets every condition in Definition 5.

## 4 State, overlay and reactions

This section illustrates how a bigraph is distributed between a processes family and how it is maintained and updated. Firstly, we formalize the idea of a "bigraph being distributed" and show how a partition of the system global state defines a semantic overlay network. The rôle of this network is crucial for the embedding algorithm since communication will follow this semantic driven structure. Finally, we describe how reactions are carried out concurrently and consistently.

In the following, let $\mathsf{Proc}$ denote the family of processes forming the distributed machine under definition and let $G$ be a generic concrete bigraph $(V_G, E_G, \mathrm{ctrl}_G, \mathrm{prnt}_G, \mathrm{link}_G) : \langle m_G, X_G\rangle \to \langle n_G, Y_G\rangle$ over a given signature $\Sigma$.

*State partition* Intuitively, a partition of the shared state $G$ is a map assigning each component of the bigraph $G$ to the process in charge of maintaining it.

**Definition 6 (State partition).** *A partition of (the shared state) $G$ over* $\mathsf{Proc}$ *is a map $\mathbb{P} : G \to$* $\mathsf{Proc}$ *assigning each component of $G$ to some process. In particular, $\mathbb{P}$ is given by the (sub)maps $\mathbb{P}^{\mathsf{v}}$, $\mathbb{P}^{\mathsf{e}}$, $\mathbb{P}^{\mathsf{s}}$, $\mathbb{P}^{\mathsf{r}}$, $\mathbb{P}^{\mathsf{i}}$, and $\mathbb{P}^{\mathsf{o}}$ on vertices, edges, sites, roots, inner names, and outer names respectively. Every component of $G$ in the pre-image of a process is said to be* held *by that process. Ports are mapped into the process holding their node i.e. $\mathbb{P}((v,i)) \triangleq \mathbb{P}(v)$.*

Then, the notion of adjacency for bigraph components can be lifted to the family of processes along the given partition map. Here hyper-edges of the link graph are considered as trees where the root and leaves are the hyper-edge handle (i.e. edge or outer name) and all the points (i.e. ports or inner names) it connects.

**Definition 7 (Adjacent processes).** *Let $R, S \in$* $\mathsf{Proc}$. *The process $R$ is said to be* adjacent (w.r.t. the partition $\mathbb{P}$) to *$S$ whenever one of the following holds:*

**(ADJ-P)** *there exists a node or site $c$ s.t. $\mathbb{P}(c) = R$ and $\mathbb{P}(\mathrm{prnt}_G(c)) = S$;*
**(ADJ-L)** *there exists a point $p$ s.t. $\mathbb{P}(p) = R$ and $\mathbb{P}(\mathrm{link}_G(p)) = S$;*
**(ADJ-R)** *there exist two roots $r, r'$ s.t. $\mathbb{P}(r) = R$ and $\mathbb{P}(r') = S$;*
**(ADJ-H)** *there exist two handles $h, h'$ s.t. $\mathbb{P}(h) = R$ and $\mathbb{P}(h') = S$.*

*In virtue of the adjacency being a symmetric relation, we will denote pairs of adjacent processes by $R \overset{\mathbb{P}}{\multimapboth} S$ and drop the partition when confusion seems unlikely. Two (partial) embeddings or a process and a (partial) embedding are said to be adjacent whenever their images are. The notation is extended accordingly.*

The adjacency relation defines an undirected graph with vertices in $\mathsf{Proc}$ and hence an *overlay network* $\mathsf{N}_{\mathbb{P}}$. The overlay network bares a specific semantic meaning since it reflects the adjacency of the bigraphical elements held by the processes forming the network: two processes are adjacent if, and only if, they hold components of the shared bigraphs $G$ that are adjacent in $G$. Moreover, for any two components of $G$, say $c_1$ and $c_2$, the shortest path in the overlay $\mathsf{N}_{\mathbb{P}}$ between the processes $\mathbb{P}(c_1)$ and $\mathbb{P}(c_2)$ will never be greater than the shortest path between $c_1$ and $c_2$ in $G$. The last observation is crucial to our purposes since relates routing through the overlay $\mathsf{N}_{\mathbb{P}}$ with walks and visits of $G$ used e.g. to compute embeddings into $G$ in non-distributed settings. Notice that the restriction of $\mathsf{N}_{\mathbb{P}}$ to $\mathrm{img}(\mathbb{P})$ will always be connected.

*Distributed reactions* Let $\phi$ be an embedding of $G$ into the bigraph shared by the process in the system and let $r : G \multimap G'$ be a parametric rewriting rule for the given BRS. Processes holding elements of $G$ image through $\phi$ or in its parameters have to negotiate the firing of $r$ and coordinate the update of their state. The negotiation phase is related to the specific execution policy and hence is left out from the present work. The update phase involves a distributed transaction and can be easily handled by established algorithms like *two-phase-commit* [5]. The embedding $\phi$ is selected among those published by the embedding engine however, the distributed transaction is still necessary since this collection of embeddings may be out of sync because of communication delays (cf. Section 5).

## 5 Distributed embedding

In this Section we present the main result of the paper: a decentralized algorithm for computing bigraphical embeddings in the distributed settings outlined in Section 4. Intuitively, each process running this algorithm maintains a collection of partial embeddings for the guests it has to look for and cooperates with its neighbouring processes (adjacency is lifted from bigraphs to processes) to complete of refute them. For the sake of simplicity we assume that all processes are given the same set of guests (e.g. the redexes of the rules of the underlying BRS) and that this set is fixed over the time; however, the algorithm can be readily adapted to work without these assumptions.

*Process structure* Each process maintain, for each guest $G$, a suitable structure $\Gamma_G$ where it stores the all partial embeddings of $G$ involving its partial view of the shared bigraph. Among these, there are all the total embeddings the process believes available at a current time and which are exposed to the outside system (e.g. the rewriting engine of the distributed bigraphical machine). Partial embeddings are decorated with some extra information to handle the non-monotonic changes of this structure over the life of the process.

**Definition 8 ($\Gamma_G$).** *A model $\Gamma$ for the guest $G$ is a set of triple $(\phi, B, ts)$ where:*

- *$\phi$ is a partial embedding from $G$ to the shared bigraph $H$;*
- *$ts$ is a logical timestamp composed by the values of the logical clocks of the processes involved in the making of $\phi$ i.e. those in $\mathrm{img}(\mathbb{P} \circ \phi)$;*
- *$B$: is a boolean value that states if $\phi$ holds. This is used to implement, together with $ts$, non-monotonic reasoning (with retracted embeddings).*

*Each model $\Gamma_G$ maintains only the last (according to the function $ts$) iteration of every partial embedding $\phi$. For this reason, we will also sometimes use $\Gamma_G$ as a partial function from partial embedding to pair $(Bool, \mathsf{Proc} \to \mathbb{N})$, s.t.:*

$$\Gamma_G(\phi) = (B, ts) \overset{\triangle}{\iff} (\phi, B, ts) \in \Gamma_G.$$

*Finally, we will say that: $\Gamma_G \models \phi \overset{\triangle}{\iff} \exists ts.(\phi, \mathtt{true}, ts) \in \Gamma_G(\phi)$.*

The procedure onBigraphViewChanged of a process $P$ is called whenever the portion of the global bigraph held by $P$ is modified. Updates define ticks in the logical clock *P.time* held by each process. Moreover, updates may invalidate some of the (partial) embeddings computed so far by the process and render new embeddings available. The first have to be retracted and the seconds have to be suggested to the nearby processes. Clearly, processes can see directly only the side effects of updates on embeddings that are "local" to them.

**Definition 9 (Local embedding).** *Let $\phi : G \hookrightarrow H$ be a partial embedding and let $\mathbb{P} : H \to \mathsf{Proc}$ be a partition. The* owners *of $\phi$ are the processes in* $\mathrm{img}(\mathbb{P}\circ\phi)$. *If $\phi$ has exactly one owner then it is said to be* local *to it. We denote the restriction of $\phi$ to the portion of bigraph held by a set of processes $S$ by $\phi\big|_{S}^{\mathbb{P}}$.*

Local embeddings can be easily computed by the algorithm proposed in [14] with minor modifications to relax the constraints ensuring totality.

Given a process $Q$, every partial embedding $\psi \sqsubseteq \phi\big|_{\{Q\}}^{\mathbb{P}}$ is local to $Q$ except for the undefined embedding – since $\mathrm{img}(\mathbb{P} \circ \varnothing)$ will always be empty. Therefore, the restriction of $\phi$ to $Q$ can be read as the largest embedding local to $Q$ that *supports* $\phi$ and every change in the state held by $Q$ that invalidates this local embedding invalidates also $\phi$ and hence have to be notified to every process owning $\phi$.

It should be noted that, before sending a local embedding (suggest), the process will check for local embeddings that were found in a previous iteration of onBigraphViewChanged and do not appear in the current one: these embeddings

```
Procedure onBigraphViewChanged()
    $time \leftarrow time + 1$
    for $G \in Guests$ do
        $localEmbeddingsOfG \leftarrow$ getLocalEmbeddings($G$)
        foreach $(\phi, \texttt{true}, ts) \in \Gamma_G$ s.t. $|\mathrm{dom}(ts)| = 1$ and
        $\phi \notin localEmbeddingsOfG$ do
            send $\langle \phi, \texttt{false}, \texttt{self} \mapsto time \rangle$ to self // self retraction
        end
        foreach $\phi \in localEmbeddingsOfG$ s.t. $\Gamma_G \not\models \phi$ do
            send $\langle \phi, \texttt{true}, \texttt{self} \mapsto time \rangle$ to self // self suggestion
        end
    end
```

are now erroneous and hence retracted by the process. Notice that the process will never send the empty embedding.

In both cases, the message sent is an entry of a model $\Gamma$ for a guest $G$, where the timestamp is the partial function defined only on the process holding the partial embedding (hence a pair $P \mapsto time$) and the boolean value included in the message is used to tell *retract* and *suggest* messages apart. These informations offer us a causal ordering between suggestions and retractions and hence the ability to handle non-monotonic reasoning in a system where messages can be received out of order. In our system, a process can compute an embedding $\phi$ that cannot be used in a reaction, for example if a retracting message about an embedding $\psi \sqsubseteq \phi$ has not been received yet. However, system consistency will be preserved because rewritings are performed inside distributed transitions and hence at least one of the processes that retracted $\phi$ will abort the transaction.

*Retracts* onBigraphViewChanged is the only procedure that generate retracted embeddings. Here we will explain why we only need to retract *local embeddings*. After a reaction, each process involved can decide that some embeddings no longer apply. Each process has only a limited knowledge about the system's bigraph, given by the portion assigned to it: for this reason, a process can only see the untruth of an embedding if it maps elements of the guest to elements assigned to that process. More formally, the set of embeddings that the process $P$ can see as false can be written as $\mathcal{R}_P \subseteq \{\phi \mid P \in \mathrm{img}(\mathbb{P} \circ \phi)\}$.

For each embedding $\phi$, if a process $P$ is involved in its formation, then there exists at least one local embedding $\psi$ computed by $P$ such that $\psi \sqsubseteq \phi$. Given its local knowledge about the global bigraph, we can conclude that if $\phi \in \mathcal{R}_P$, then there exists $\psi \sqsubseteq \phi$ local to $P$ and such that $\psi \in \mathcal{R}_P$. Therefore, if a process $P$ wants to retract each embedding in $\mathcal{R}_P$, it only needs to retract the subset of its local embeddings $\{\phi \mid \phi \in \mathcal{R}_P \wedge \{P\} = \mathrm{img}(\mathbb{P} \circ \phi)\}$.

*$\Gamma$-updates* When a process $P$ receives a *retraction message* $\langle \phi, \texttt{false}, ts \rangle$[1] such that $\phi$ occurs in $\Gamma$ but the occurrence was generated earlier than $ts$ it invali-

[1] Notice that the partial embedding being retracted is local.

```
Procedure retract($G$,$\phi$,$ts$)
    // $ts$ involves exactly one process
    $\{P\} \leftarrow \mathrm{dom}(ts)$
    $t' \leftarrow 0$
    if $\mathit{\Gamma}_G(\phi) \neq \bot$ then // new embedding
        $(B', ts') \leftarrow \mathit{\Gamma}_G(\phi)$
        $t' \leftarrow ts'(P)$
    end
    if $ts(P) > t'$ then
        $D \leftarrow \emptyset$
        foreach $(\psi, B'', ts'') \in \mathit{\Gamma}_G$ do
            if $\phi \sqsubseteq \psi \wedge t > ts''(P)$ then // $\frown$ relation
                $ts''(P) \leftarrow t$
                $\mathit{\Gamma}_G(\psi) \leftarrow (\mathtt{false}, ts'')$
                $D \leftarrow D \cup \{P \mid \mathtt{self} \multimapboth P \wedge P \in \mathrm{img}(\mathbb{P} \circ \psi)\}$
            end
        end
        send $\langle \phi, \mathtt{false}, ts \rangle$ to $D$
    end
```

dates every $\psi$ in $\mathit{\Gamma}$ made from $\phi$ and more recent than $ts$ and then forwards the retraction message to every neighbour process involved by the retraction i.e. appearing in a partial embedding being removed from $\mathit{\Gamma}$. Formally, we define a *retract relation* $\frown$ between local and partial embedding:

$$(\phi, P \mapsto \mathrm{t}) \frown (\psi, ts) \overset{\triangle}{\iff} \phi \sqsubseteq \psi \wedge t > ts(P)$$

With this relation, we can define the first updating rule for $\mathit{\Gamma}_G$ as follows:

$$\frac{(\psi, B, ts) \in \mathit{\Gamma}_G \qquad (\phi, P \mapsto \mathrm{t}) \frown (\psi, ts) \qquad (\psi, \mathtt{false}, ts') \in \mathit{\Gamma}'_G}{\mathit{\Gamma}_G \xrightarrow{\langle \phi, \mathtt{false}, P \mapsto t \rangle} \mathit{\Gamma}'_R} \qquad (\mathit{\Gamma}\text{-UP1})$$

where

$$ts'(x) \triangleq \begin{cases} t, & \text{if } x = P. \\ ts, & \text{otherwise.} \end{cases}$$

and $\mathit{\Gamma}'_G$ it's equal to $\mathit{\Gamma}_G$ for each $(\psi, B, ts) \in \mathit{\Gamma}_G$ s.t. $(\phi, P \mapsto \mathit{time}) \not\frown (\psi, ts)$. The procedure retract implements what we have seen so far in this section, and it models rather closely the updating rule ($\mathit{\Gamma}$-UP1).

If, instead, $P$ receives a *suggestion message* $\langle \phi, \mathtt{true}, ts \rangle$, it needs to update a subset of its embeddings and derive new embeddings. These two actions are implemented respectively by the procedures suggest and combine. Embeddings that need to be updated are all $\psi$ in $\mathit{\Gamma}_G$, $\mathit{\Gamma}_G(\psi) = (B', ts')$, such that $(\phi, ts) \smile (\psi, B', ts')$, where the *update relation* $\smile$ is defined as follows:

$$\begin{aligned}(\phi, ts) \smile (\psi, B', ts') \overset{\triangle}{\iff} \psi \sqsubseteq \phi \wedge \forall P. ts'(P) \leq ts(P) \wedge \\ (B' \implies \exists Q. ts'(Q) < ts(Q))\end{aligned}$$

```
Procedure suggest($G$,$\phi$,$ts$)
    if $\Gamma_G(\phi) = \bot$ then // new embedding
        $\Gamma_G(\phi) \leftarrow (\mathtt{true}, ts)$
        send $\langle \phi, \mathtt{true}, ts \rangle$ to $\{P \mid \phi \multimapboth P\}$
        combine($G$,$\phi$,$ts$)
    end
    foreach $(\psi, B', ts') \in \Gamma_G$ do
        // $\smile$ relation
        if $\psi \sqsubseteq \phi \wedge \forall (P,t) \in ts'\ t \leq ts(P)\ \wedge\ (B' \rightarrow \exists (P,t) \in ts'\ t < ts(P))$ then
            $ts' \leftarrow \{(P,t) | (P,t) \in ts \wedge \exists t' (P,t') \in ts'\}$
            $\Gamma_G(\psi) \leftarrow (\mathtt{true}, ts')$
            send $\langle \psi, \mathtt{true}, ts' \rangle$ to $\{P \mid \psi \multimapboth P\}$
            combine($G$,$\psi$,$ts'$)
        end
    end
```

An embedding $\psi \sqsubseteq \phi$ in $\Gamma_G$ needs to be updated if the time associated to each process via the logical timestamp stored in $\Gamma_G(\psi)$ is lower or equal than its counterpart in the message's timestamp $ts$. Also, if $\Gamma_G \models \psi$, $\psi$ will only be updated if exists a process $P$ such that $ts(P)$ is strictly greater than the time of $P$ associated with $\psi$ in $\Gamma_G$. This last constraint ensures that if a process receives multiple instances of the same *suggesting* message, it will not update $\Gamma_G$ and send that message to its neighbourhood more than once. We can now defines the $\Gamma_G$-update rule for suggesting messages:

$$\frac{(\psi, B', ts') \in \Gamma_G \quad (\phi, ts) \smile (\psi, B', ts') \quad (\psi, true, ts'') \in \Gamma'_R}{\Gamma_G \xrightarrow{(G,\phi,\mathtt{true},ts)} \Gamma'_R} \qquad (\Gamma\text{-UP2})$$

where $ts'' = ts\big|_{\mathrm{dom}(ts')}$.

After this update step, each updated embedding will be used to derive new embeddings. Given an updated embedding $\phi$ from the previous step, processes will search for all $\psi$ such that $\phi \sqcup_{\downarrow} \psi$, where the relation $\sqcup_{\downarrow}$, between partial embeddings, is defined as follows:

$$\phi \sqcup_{\downarrow} \psi \overset{\triangle}{\iff} \psi \not\sqsubseteq \phi \wedge \phi \not\sqsubseteq \psi \wedge \phi \sqcup \psi \text{ is a partial embedding}$$

We can now define the third updating rule for $\Gamma_G$, which describes how embeddings are derived:

$$\frac{(\phi, \mathtt{true}, ts) \in \Gamma_G \qquad (\psi, \mathtt{true}, ts') \in \Gamma_G \qquad \phi \sqcup_{\downarrow} \psi}{(\phi \sqcup \psi, \mathtt{true}, ts'') \in \Gamma_G} \qquad (\Gamma\text{-UP3})$$

where

$$P''(x) = \begin{cases} \max(P(x), P'(x)), & \text{if } \overline{\Gamma}_R \not\models \phi \sqcup \psi; \\ \max(P(x), P'(x), \pi_2(\overline{\Gamma}_R(\phi \sqcup \psi))(x)), & \text{otherwise.} \end{cases}$$

```
Procedure combine($G$,$\phi$,$ts$)
    foreach $(\psi, B', ts') \in \Gamma_G$ do
        $\rho \leftarrow \phi \sqcup \psi$
        if $B' \wedge \psi \not\sqsubseteq \phi \wedge \phi \not\sqsubseteq \psi \wedge$ isConsistent($\rho$) then // $\sqcup_\downarrow$ relation
            $ts'' \leftarrow \{(P,t) \mid (ts(P) \neq \bot \vee ts'(P) \neq \bot) \wedge t = \max(ts(P), ts'(P))\}$
            if $\Gamma_G(\rho) = \bot$ then
                $\Gamma_G(\rho) \leftarrow$ (true, $ts''$)
                send $\langle \rho$, true, $ts'' \rangle$ to $\{P \mid \rho \multimap P\}$
            else
                ($oldB$,$oldts$) $\leftarrow \Gamma_G(\rho)$
                if $oldB$ then
                    $ts'' \leftarrow \{(P, \max(t_1, t_2)) \mid (P, t_1) \in ts'' \wedge (P, t_2) \in oldts\}$
                end
                if $\forall (P,t) \in oldts\ t \leq ts''(P)$ then
                    $\Gamma_G(\rho) \leftarrow$ (true, $ts''$)
                    send $\langle \rho$, true, $ts'' \rangle$ to $\{P \mid \rho \multimap P\}$
                end
            end
        end
    end
```

and $\overline{\Gamma}_R$ is $\Gamma_G$ before the new derivation of $\phi \sqcup \psi$ (with the previous timestamp for $\phi \sqcup \psi$). This updating rule, if abstracted from the timestamps $ts$ and $ts'$, can be expressed the following cleaner form:

$$\frac{\Gamma_G \models \phi \qquad \Gamma_G \models \psi \qquad \phi \sqcup_\downarrow \psi}{\Gamma_G \models \psi \sqcup \phi}$$

Each embedding updated or derived from a suggestion message will update $\Gamma$ and will be sent to the process neighbourhood restricted to those processes that are considered adjacent to the embedding i.e. those holding some component of the shared bigraph that is adjacent (in the sense of Definition 7) to the image of the partial embedding. Embeddings that are completed are exposed to the outer system contextually to the update of $\Gamma$.

To make our set of updating rules complete, we need two additional rules to manage incoming messages carrying embeddings that were never seen before (i.e. $\Gamma_G(\phi) = \bot$). This two rules are implemented by retract and suggest procedures with minor changes w.r.t. the case of ($\Gamma$-UP1) and ($\Gamma$-UP2).

## 6 Conclusions and future work

In this paper we have presented an algorithm for computing bigraph embeddings in a distributed environment where the host bigraph is spread across several co-operating processes. Differently from existing algorithms [8,14,21], this algorithm is completely decentralized and does not require any process in the system to have a complete view of the global state, hence it can scale to handle bigraphs too large to reside in the memory of a single process/machine. Moreover, embeddings that are not affected by a reaction are not recomputed and in general the computation of an embedding requires a number of messages that is linearly bounded by the size of the embedded bigraph. However, the overall network impact is not negligible and, in the worst case, can be outperformed by the "semi-distributed" algorithm proposed in [12] where processes visit the shared bigraph (the visit is guaranteed to be minimal by the use of IPOs) and compute embeddings locally using the information gathered. Fortunately there is room for improvement for the algorithm proposed: suggestion and retraction messages can be grouped and compressed by suitable representations since the combinatoric explosion is due to symmetries and isomorphisms between local partial embeddings. Moreover, symbolic representations can be put in place to further reduce the communication footprint of the algorithm. We leave this developments for the extended version of the paper and future works.

The distributed embedding algorithm is the basic block of the *distributed bigraphical machine*, a distributed instance of the abstract bigraph machine. This machine inherits the benefits of the decentralized algorithm, e.g. its scalability.

A direct application of the distributed embedding algorithm is to simulate, or execute, multi-agent systems. In [12] the authors devise a methodology for design and prototype multi-agent systems with BRS. Intuitively, the application domain is modelled by a BRS and entities in its states are divided as "subjects" and "objects" depending on their ability to actively perform actions. Subjects are precisely the agents of the system and reactions are reconfigurations. This observation yields a coherent way to partition and distribute a bigraph among the agents, which can be assimilated to the processes of the distributed bigraphical machine (execution policies are defined by agents desires and goals). Therefore, these agents can find and perform bigraph rewritings in a truly concurrent, distributed fashion, by using the distributed embedding algorithm.

How the bigraph is partitioned and distributed can affect the performance of the system. For instance, it is easy to devise a situation in which even relatively small guests require the cooperation of several processes, say nearly one for each component of the guest. An interesting line of research would be to study the relation between guests, partitions, and performance in order to develop efficient distribution strategies. Moreover, structured partitions lend themselves to ad-hoc heuristics and optimizations. As an example, the way bigraphs are distributed among agents in [12] takes into account how they interact and reconfigure.

We considered adjacency as an undirected graph but some information is lost in this simplification. In fact, place and link graphs can be seen as forests suggesting the use of directed graphs. We intend to use this additional information to improve the routing through the induced semantic (directed) network.

## References

1. G. Bacci, D. Grohmann, and M. Miculan. Bigraphical models for protein and membrane interactions. In G. Ciobanu, editor, *Proc. MeCBIC*, volume 11 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–18, 2009.

2. G. Bacci, M. Miculan, and R. Rizzi. Finding a forest in a tree. In *Proc. TGC*, *Lecture Notes in Computer Science*. Springer, 2014.
3. L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical models of context-aware systems. In L. Aceto and A. Ingólfsdóttir, editors, *Proc. FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2006.
4. M. Bundgaard, A. J. Glenstrup, T. T. Hildebrandt, E. Højsgaard, and H. Niss. Formalizing higher-order mobile embedded business processes with binding bigraphs. In D. Lea and G. Zavattaro, editors, *COORDINATION*, volume 5052 of *Lecture Notes in Computer Science*, pages 83–99. Springer, 2008.
5. E. C. Cooper. Analysis of distributed commit protocols. In *Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, pages 175–183. ACM, 1982.
6. T. C. Damgaard, E. Højsgaard, and J. Krivine. Formal cellular machinery. *Electronic Notes in Theoretical Computer Science*, 284:55–74, 2012.
7. A. J. Faithfull, G. Perrone, and T. T. Hildebrandt. BigRed: A development environment for bigraphs. *ECEASST*, 61, 2013.
8. A. Glenstrup, T. Damgaard, L. Birkedal, and E. Højsgaard. An implementation of bigraph matching. *IT University of Copenhagen*, 2007.
9. E. Højsgaard. *Bigraphical Languages and their Simulation.* PhD thesis, IT University of Copenhagen, 2012.
10. O. H. Jensen and R. Milner. Bigraphs and transitions. In A. Aiken and G. Morrisett, editors, *POPL*, pages 38–49. ACM, 2003.
11. J. Krivine, R. Milner, and A. Troina. Stochastic bigraphs. In *Proc. MFPS*, volume 218 of *Electronic Notes in Theoretical Computer Science*, pages 73–96, 2008.
12. A. Mansutti, M. Miculan, and M. Peressotti. Multi-agent systems design and prototyping with bigraphical reactive systems. In K. Magoutis and P. Pietzuch, editors, *DAIS*, volume 8460 of *Lecture Notes in Computer Science*, pages 201–208. Springer, 2014.
13. M. Miculan and M. Peressotti. Bigraphs reloaded. Technical Report UDMI/01/2013, Department of Mathematics and Computer Science, University of Udine, 2013.
14. M. Miculan and M. Peressotti. A CSP implementation of the bigraph embedding problem. In T. T. Hildebrandt, editor, *Proc. MeMo*, 2014.
15. R. Milner. *The Space and Motion of Communicating Agents.* Cambridge University Press, 2009.
16. E. Pereira, C. M. Kirsch, J. B. de Sousa, and R. Sengupta. BigActors: a model for structure-aware computation. In C. Lu, P. R. Kumar, and R. Stoleru, editors, *ICCPS*, pages 199–208. ACM, 2013.
17. G. Perrone, S. Debois, and T. T. Hildebrandt. Bigraphical refinement. In J. Derrick, E. A. Boiten, and S. Reeves, editors, *Proc. REFINE*, volume 55 of *Electronic Proceedings in Theoretical Computer Science*, pages 20–36, 2011.
18. G. Perrone, S. Debois, and T. T. Hildebrandt. A model checker for bigraphs. In S. Ossowski and P. Lecca, editors, *Proc. SAC*, pages 1320–1325. ACM, 2012.
19. G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation*, volume 1. World Scientific, River Edge, NJ, USA, 1997.
20. M. Sevegnani and E. Pereira. Towards a bigraphical encoding of actors. In T. T. Hildebrandt, editor, *Proc. MeMo*, 2014.
21. M. Sevegnani, C. Unsworth, and M. Calder. A SAT based algorithm for the matching problem in bigraphs with sharing. Technical Report TR-2010-311, Department of Computer Science, University of Glasgow, 2010.