

# Observational Equivalence for Synchronized Graph Rewriting with Mobility<sup>\*</sup>

Barbara König and Ugo Montanari

Dipartimento di Informatica, Università di Pisa, Italia  
{koenigb,ugo}@di.unipi.it

**Abstract.** We introduce a notion of bisimulation for graph rewriting systems, allowing us to prove observational equivalence for dynamically evolving graphs and networks.

We use the framework of synchronized graph rewriting with mobility which we describe in two different, but operationally equivalent ways: on graphs defined as syntactic judgements and by using tile logic. One of the main results of the paper says that bisimilarity for synchronized graph rewriting is a congruence whenever the rewriting rules satisfy the basic source property. Furthermore we introduce an up-to technique simplifying bisimilarity proofs and use it in an example to show the equivalence of a communication network and its specification.

## 1 Introduction

Graph rewriting can be seen as a general framework in which to specify and reason about concurrent and distributed systems [8]. The topology and connection structure of these systems can often be naturally represented in terms of nodes and connecting edges, and their dynamic evolution can be expressed by graph rewriting rules. We are specifically interested in hypergraphs where an arbitrarily long sequence of nodes—instead of a pair of nodes—is assigned to every edge.

However, the theory of graph rewriting [24] lacks a concept of observational equivalence, relating graphs which behave the same in all possible context, which is quite surprising, since observational equivalences, such as bisimilarity or trace equivalence, are a standard tool in the theory of process calculi.

We are therefore looking for a semantics for (hyper-)graph rewriting systems that abstracts from the topology of a graph, and regards graphs as processes which are determined by their interaction with the environment, rather than by their internal structure. It is important for the observational equivalence to be a congruence, since this will enable compositional proofs of equivalence and assure substitutivity, i.e. that equivalent subcomponents of a system are exchangeable.

The applications we have in mind are the verification of evolving networks, consisting, e.g., of processes, messages and other components. A possible scenario

---

<sup>\*</sup> Research partly supported by TMR Network GETGRATS, by Esprit WG APPLI-GRAPH and by MURST project TOSCA.

would be a user who has limited access to a dynamically changing network. We want to show that the network is transparent with respect to the location of resources, failure of components, etc. by showing that it is equivalent to a simpler specification. Such an equivalence of networks is treated in the example in Section 7.

One possible (and well-studied) candidate for an observational equivalence is bisimilarity. So the central aim of this paper is to introduce bisimilarity for graph rewriting—we will explain below why it is convenient to base this equivalence on the model of synchronized graph rewriting, as opposed to other models—and to introduce an up-to proof technique, simplifying actual proofs of bisimilarity.

When defining bisimulation and bisimilarity for graph rewriting systems, two possibilities come to mind: the first would be to use unlabelled context-sensitive rewrite rules as, for example, in the double-pushout approach [7]. The definition of an observational congruence in this context, however, ordinarily requires universal quantification over all possible contexts of an expression, which is difficult to handle in practice. This makes us choose the second possibility, which is closer to process algebras: we use synchronized graph rewriting, which allows only context-free rewrite rules whose expressive power is increased considerably by adding synchronization and mobility (i.e. communication of nodes), thus including a large class of rewriting systems. In this case we can define a simple syntactic condition (the basic source property) on the rewrite rules ensuring that bisimilarity is a congruence (compare with the de Simone format [5] and the `tyft/tyxt`-format [12]).

As synchronization mechanism we choose Hoare synchronization which means that all edges that synchronize via a specific node produce the same synchronization action. This is different from Milner synchronization (as in CCS [19]) where two synchronizing processes produce two different signals: an action  $a$  and a coaction  $\bar{a}$ .

We prefer Hoare synchronization since it makes it easier to handle the kind of multiple synchronization we have in mind: several edges connected to each other on a node must agree on an action  $a$ , which means that there is no clear distinction between action and coaction. This, in turn, causes other nodes connected to the same edges to perform a different action, and in this way synchronization is propagated by edges and spreads throughout an entire connected component. It is conceivable to implement different synchronization mechanisms as processes working as “connectors”, thus modeling in this way a variety of coordination mechanisms.

Edges synchronizing with respect to an action  $a$  may, at the same time, agree to create new nodes which are then shared among the right-hand sides of the respective rewrite rules. (This form of mobility was first presented in [13] and is also extensively treated in [14].) From the outside it is not possible to determine whether newly created nodes are different or equal, it is only possible to observe the actions performed.

Apart from the obvious representation of graphs in terms of nodes and edges, there are several other approaches representing graphs by terms, which allow for

a more compositional presentation of graphs and enable us, for example, to do induction on the graph structure. We will introduce two of these term representations: first graphs as syntactic judgements, where nodes are represented by names and we have operators such as parallel composition and name hiding at our disposal. This representation allows for a straightforward definition of graph rewriting with synchronization and mobility.

The second representation defines graphs in terms of arrows of a P-monoidal category [3]. This allows for an easy presentation of graph rewriting in tile logic, a rewriting framework which deals with the rewriting of open terms that can still be contextualized and instantiated and allows for different ways of composing partial rewrites. To show the compositionality of our semantics, we use a property of tile logic, i.e. the fact that if a tile system satisfies a so-called decomposition property, then bisimilarity defined on top of this tile system is a congruence (see also [1]).

Apart from the fact that we use tile logic as a tool to obtain the congruence result, we also show how mobility, and specifically the form of mobility used in synchronized graph rewriting, can be handled in the context of tile logic.

## 2 Synchronized Graph Rewriting with Mobility

We start by introducing a representation of (hyper-)graphs as syntactic judgements, where nodes in general correspond to names, external nodes to free names and (hyper-)edges to terms of the form  $s(x_1, \dots, x_n)$  where the  $x_i$  are arbitrary names.

**Definition 1 (Graphs as Syntactic Judgements).** *Let  $N$  be a fixed infinite set of names. A syntactic judgement is of the form  $\Gamma \vdash G$  where  $\Gamma \subseteq N$  is a set of names (the interface of the graph) and  $G$  is generated by the grammar*

$$G ::= \text{nil (empty graph)} \mid G|G \text{ (parallel composition)} \mid \\ (\nu x)G \text{ (node hiding)} \mid s(x_1, \dots, x_n) \text{ (edge)}$$

where  $x \in N$  and  $s(x_1, \dots, x_n)$  with arbitrary  $x_i \in N$  is called an edge of arity  $n$  labelled  $s$ . (Every label is associated with a fixed arity.)

Let  $\text{fn}(G)$  denote the set of all free names of  $G$ , i.e. all names not bound by a  $\nu$ -operator. We demand that  $\text{fn}(G) \subseteq \Gamma$ .

We assume that whenever we write  $\Gamma, x$ , then  $x$  is not an element of  $\Gamma$ .

We need to define a structural congruence on syntactic judgements in order to identify those terms that represent isomorphic graphs (up to isolated nodes) (see [15, 16]).

**Definition 2 (Structural Congruence).** *Structural congruence  $\equiv$  on syntactic judgements obeys the rules below and is closed under parallel composition  $|$  and the hiding operator  $\nu$ . (We abbreviate equations of the form  $\Gamma \vdash G \equiv \Gamma \vdash G'$  by  $G \equiv G'$ .)*

$$\begin{aligned}
& \Gamma \vdash G \equiv \rho(\Gamma) \vdash \rho(G) \text{ where } \rho \text{ is an injective substitution} \\
& (G_1|G_2)|G_3 \equiv G_1|(G_2|G_3) \quad G_1|G_2 \equiv G_2|G_1 \quad G|nil \equiv G \\
& (\nu x)(\nu y)G \equiv (\nu y)(\nu x)G \quad (\nu x)nil \equiv nil \quad (\nu x)G \equiv (\nu y)G\{y/x\} \text{ if } y \notin fn(G) \\
& (\nu x)(G|G') \equiv (\nu x)G|G' \text{ if } x \notin fn(G')
\end{aligned}$$

We sometimes abbreviate  $(\nu x_1) \dots (\nu x_n)G$  by  $(\nu\{x_1, \dots, x_n\})G$ .

*Example 1.* We regard the syntactic judgement  $y \vdash (\nu x)(\nu z)(P(x) | S(x, y, z) | P(z))$  which consists of two processes  $P$  which are connected to each other and the only external node  $y$  via a switch  $S$ . A graphical representation of this syntactic judgement can be found in Figure 2 (graph in the lower left corner).

In order to define rewriting on syntactic judgements we introduce the notion of rewriting rule. We use a set  $Act$  of arbitrary actions, which can be thought of as the set of signals which are allowed in a network.

**Definition 3 (Rewriting Rules).** *Let  $Act$  be a set of actions, containing also the idle action  $\varepsilon$ . Each action  $a \in Act$  is associated with an arity  $ar(a) \in \mathbb{N}$ , the arity of  $\varepsilon$  is 0. (The arity indicates the number of nodes created by an action.)*

*A rewriting rule is of the form*

$$x_1, \dots, x_n \vdash s(x_1, \dots, x_n) \xrightarrow{\Lambda} x_1, \dots, x_n, \Gamma_\Lambda \vdash G$$

where all  $x_i$  are distinct,  $\Lambda \subseteq \{x_1, \dots, x_n\} \times Act \setminus \{\varepsilon\} \times \mathbb{N}^*$  such that  $\Lambda$  is a total function in its first argument, i.e. if  $(x_i, a_i, \tilde{y}_i) \in \Lambda$  we write  $\Lambda(x_i) = (a_i, \tilde{y}_i)$ , respectively  $act_\Lambda(x_i) = a_i$  and  $n_\Lambda(x_i) = \tilde{y}_i$ , and we demand that  $ar(a_i) = |\tilde{y}_i|$ .

Furthermore<sup>1</sup>  $\Gamma_\Lambda = \bigcup_{x_i \in \Lambda} Set(n_\Lambda(x_i))$  and we demand that  $\{x_1, \dots, x_n\} \cap \Gamma_\Lambda = \emptyset$ .

A rewriting rule of the form given above indicates that an edge  $s(x_1, \dots, x_n)$  is rewritten, synchronizing on each node  $x_i$  with an action  $a_i$ , and during this synchronization a string  $\tilde{y}_i$  of new nodes is created. The set  $\Gamma_\Lambda$  contains all new nodes in the interface which are created by the rewriting step.

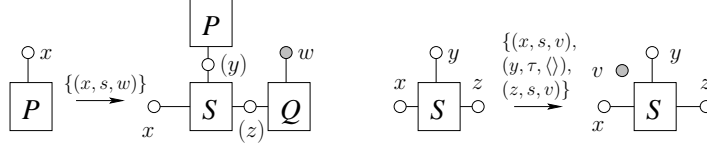
The following example will be used as a running example throughout the paper.

*Example 2.* We describe a network of processes  $P$  of arity 1 and processes  $Q$  of arity 2 connected to each other via switches  $S$  of arity 3.

We use three kinds of actions, apart from the idle action  $\varepsilon$ :  $\tau$  and  $a$  (both of arity 0) and  $s$  (of arity 1) which is the action used for establishing a shared name. A process of our example network can perform the following rewriting steps:<sup>2</sup>  $P$  can either send a signal  $a$ , or it can extend the network by transforming itself into a switch with two processes connected to it, or it can perform an  $s$  action and fork a process  $Q$  whose second node is connected to a newly created, privately

<sup>1</sup> For any string  $\tilde{s}$ , we denote the set of its elements by  $Set(\tilde{s})$ .

<sup>2</sup> The empty sequence is denoted by  $\langle \rangle$ .



**Fig. 1.** Graphical representation of example rules

shared channel. The action  $\tau$  is different from the idle action and is used in this example to represent internal activity.

$$\begin{aligned}
& x \vdash P(x) \xrightarrow{\{(x, a, \langle \rangle)\}} x \vdash P(x) \\
& y \vdash P(y) \xrightarrow{\{(y, \tau, \langle \rangle)\}} y \vdash (\nu x)(\nu z)(P(x) \mid S(x, y, z) \mid P(z)) \\
& x \vdash P(x) \xrightarrow{\{(x, s, w)\}} x, w \vdash (\nu y)(\nu z)(S(x, y, z) \mid P(y) \mid Q(z, w)).
\end{aligned}$$

The process  $Q$ , on the other hand, can perform any combination of  $a$  and  $\tau$  actions.

$$x, y \vdash Q(x, y) \xrightarrow{\{(x, a_1, \langle \rangle), (y, a_2, \langle \rangle)\}} x, y \vdash Q(x, y) \quad \text{where } a_1, a_2 \in \{a, \tau\}.$$

Switches have the task to route the signals and actions originating at the processes and in the case of an  $s$  action a new node  $v$  is created. In both rules we require that  $\{x, y, z\} = \{x_1, x_2, x_3\}$ :

$$\begin{aligned}
& x, y, z \vdash S(x, y, z) \xrightarrow{\{(x_1, a, \langle \rangle), (x_2, a, \langle \rangle), (x_3, \tau, \langle \rangle)\}} x, y, z \vdash S(x, y, z) \\
& x, y, z \vdash S(x, y, z) \xrightarrow{\{(x_1, s, v), (x_2, s, v), (x_3, \tau, \langle \rangle)\}} x, y, z, v \vdash S(x, y, z)
\end{aligned}$$

A graphical representation of the third rule for  $P$  and the second rule for  $S$  (with  $x_1 = x, x_2 = z, x_3 = y$ ) is depicted in Figure 1, where the bound names are indicated by their enclosure in round brackets.

In order to be able to define inference rules which describe how to derive more complex transitions from the basic rules, we first introduce the following notion of a most general unifier which transforms a relation  $\Lambda$ , which does not necessarily satisfy the conditions of definition 3, into a function.

**Definition 4 (Most General Unifier).** *Let  $\sigma : N \rightarrow N$  be a name substitution. If  $\Lambda = \{(x_i, a_i, \tilde{y}_i) \mid i \in \{1, \dots, n\}\}$ , then  $\sigma(\Lambda) = \{(\sigma(x_i), a_i, \sigma^*(\tilde{y}_i)) \mid i \in \{1, \dots, n\}\}$  where  $\sigma^*$  is the extension of  $\sigma$  to strings.*

*For any  $\Lambda = \{(x_i, a_i, \tilde{y}_i) \mid i \in \{1, \dots, n\}\} \subseteq N \times \text{Act} \times N^*$  we call a substitution  $\rho : N \rightarrow N$  a unifier of  $\Lambda$  whenever  $\rho(x_i) = x_i$  for  $i \in \{1, \dots, n\}$  and  $x_i = x_j$  implies  $a_i = a_j$  and  $\rho^*(\tilde{y}_i) = \rho^*(\tilde{y}_j)$ .*

*The mapping  $\rho$  is called a most general unifier whenever it is a unifier with a minimal degree of non-injectivity. Unifiers do not necessarily exist.*

*Example 3.* The substitution  $\rho = \{u/v, u/r, u/s, u/w, u/t\}$  is a unifier for  $\Lambda = \{(x, a, uvvw), (x, a, rsst)\}$  since  $\rho(\Lambda) = \{(x, a, uvuu)\}$ . A most general unifier is, for example,  $\rho' = \{u/v, u/r, u/s, w/t\}$  where  $\rho'(\Lambda) = \{(x, a, uvuw)\}$ .

The set  $\Lambda = \{(x, a, u), (x, b, v)\}$ , where  $a \neq b$ , does not have a unifier.

Most general unifiers are needed in order to make sure that whenever two nodes are merged, the strings of nodes created by synchronizing on them, are also merged. Regard, for example, the rewriting rules

$$x \vdash s(x) \xrightarrow{\Lambda_1 = \{(x, a, y)\}} x, y \vdash s'(x, y) \text{ and } x \vdash t(x) \xrightarrow{\Lambda_2 = \{(x, a, z)\}} x, z \vdash t'(x, z).$$

Then—since the edges  $s$  and  $t$  should agree on a common new name—we expect that

$$x \vdash s(x) \mid t(x) \xrightarrow{\Lambda = \{(x, a, y)\}} x, y \vdash s'(x, y) \mid t'(x, y)$$

where  $\Lambda$  can be obtained by applying the most general unifier to  $\Lambda_1 \cup \Lambda_2$ .

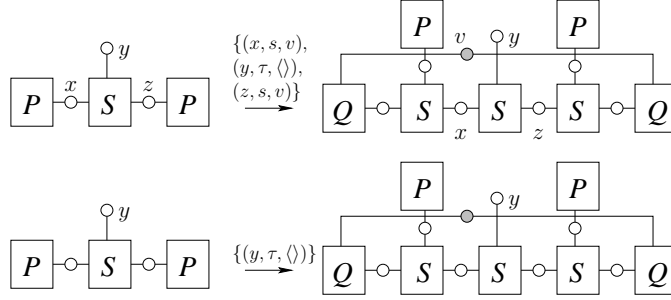
We introduce the following inference rules for transitions, which are similar to the rules given in [13, 14].

**Definition 5 (Inference Rules for Transitions).** *All possible transitions  $\Gamma \vdash G \xrightarrow{\Lambda} \Gamma' \vdash G'$  between graphs are generated by a set  $R$  of rewriting rules and the inference rules given below and are closed under injective renaming of all names occurring in a transition.*

$$\begin{aligned} (\text{ren}) \quad & \frac{\Gamma \vdash G \xrightarrow{\Lambda} \Gamma, \Gamma_\Lambda \vdash G'}{\rho(\Gamma) \vdash \rho(G) \xrightarrow{\rho'(\rho(\Lambda))} \rho(\Gamma), \Gamma_{\rho'(\rho(\Lambda))} \vdash \rho'(\rho(G'))} \\ & \text{where } \rho : \Gamma \rightarrow \Gamma \text{ and } \rho' \text{ is the most general unifier for } \rho(\Lambda). \\ (\text{par}) \quad & \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda_1} \Gamma, \Gamma_{\Lambda_1} \vdash G'_1 \quad \Gamma \vdash G_2 \xrightarrow{\Lambda_2} \Gamma, \Gamma_{\Lambda_2} \vdash G'_2}{\Gamma \vdash G_1 \mid G_2 \xrightarrow{\rho(\Lambda_1 \cup \Lambda_2)} \Gamma, \Gamma_{\rho(\Lambda_1 \cup \Lambda_2)} \vdash \rho(G'_1 \mid G'_2)} \\ & \text{if } \Gamma_{\Lambda_1} \cap \Gamma_{\Lambda_2} = \emptyset \text{ and } \rho \text{ is the most general unifier for } \Lambda_1 \cup \Lambda_2. \\ (\text{hide}) \quad & \frac{\Gamma, x \vdash G \xrightarrow{\Lambda \uplus \{(x, a, \tilde{y})\}} \Gamma, x, \Gamma_\Lambda, Y \vdash G'}{\Gamma \vdash (\nu x)G \xrightarrow{\Lambda} \Gamma, \Gamma_\Lambda \vdash (\nu x)(\nu Y)G'} \quad \text{where } Y = \text{Set}(\tilde{y}) \setminus \Gamma_\Lambda. \\ (\text{idle}) \quad & \Gamma \vdash G \xrightarrow{\Lambda} \Gamma \vdash G \quad \text{where}^3 \Lambda(x) = (\varepsilon, \langle \rangle) \text{ for } x \in \Gamma. \\ (\text{new}) \quad & \frac{\Gamma \vdash G \xrightarrow{\Lambda} \Gamma, \Gamma_\Lambda \vdash G'}{\Gamma, x \vdash G \xrightarrow{\Lambda \uplus \{(x, a, \tilde{y})\}} \Gamma, x, \Gamma_\Lambda, \tilde{y} \vdash G'} \end{aligned}$$

We also write  $R \Vdash (\Gamma \vdash G \xrightarrow{\Lambda} \Gamma' \vdash G')$  whenever this transition can be derived from a set  $R$  of rewriting rules.

<sup>3</sup> The empty sequence is denoted by  $\langle \rangle$ .



**Fig. 2.** Processes establishing a privately shared channel

In every transition  $\Lambda$  assigns to each free name the action it performs and the string of new nodes it creates. Rule *(ren)* deals with non-injective renaming of the nodes of a graph, which is necessary in order to handle edges of the form  $s(\dots, x, \dots, x, \dots)$ , i.e. edges which are connected several times to the same node. Parallel composition of syntactic judgements is treated in rule *(par)* which makes sure that whenever a synchronization on a node creates a string  $\tilde{y}_1$  in the rewriting of  $\Gamma \vdash G_1$  and the synchronization on the same node creates a string  $\tilde{y}_2$  in the rewriting of  $\Gamma \vdash G_2$ , then both strings are identified by  $\rho$ . In rule *(hide)*, which deals with hiding of names, we do not only have to hide the name itself, but all the names which have been created exclusively by interaction on this name, i.e. all names in the set  $Y$ . Furthermore every syntactic judgement can always make an explicit idle step by performing action  $\varepsilon$  on all its external nodes (rule *(idle)*) and we can add an additional name to the interface which performs arbitrary actions (rule *(new)*). This is due to Hoare synchronization which requires that any number of edges, and therefore also zero edges, can synchronize on a given node.

*Example 4.* One of the most interesting rewriting steps which can be derived from the rules of example 2 is the forking of two processes  $Q$  at the same time and the establishment of a privately shared channel between them. We intend to reduce the syntactic judgement  $y \vdash (\nu x)(\nu z)(P(x) \mid S(x, y, z) \mid P(z))$  from example 1. The task of the switch  $S$  is to route the signal  $s$  on which both processes synchronize, and also to propagate the newly created name.

We first derive a transition for  $x, y, z \vdash P(x) \mid S(x, y, z) \mid P(z)$  which is depicted in the upper half of Figure 2 and which can be obtained by composing the rewriting rules given in Figure 1 where the concept of the most general unifier forces  $v = w$ . Then in the next step we hide both names  $x$  and  $z$  which causes all names produced by interaction on  $x$  or  $z$  to be hidden as well, which means that  $v$  is removed from the interface (see the lower half of Figure 2).

We can also observe that when a process  $P$  creates a new node which is communicated to the environment, a form of name extrusion as in the  $\pi$ -calculus [21] is performed. In the extrusion rule of the labelled transition semantics of the

$\pi$ -calculus, a private, but extruded, name may also appear free in the right-hand side of the rule.

### 3 Representation of Graphs in a P-monoidal Category

In order to be able to describe graph rewriting in tile logic, we will now describe a second possibility of graph representation, which abstracts from names, i.e. nodes are not addressed via their name, but via their position in the interface. In this way we identify all graphs which can be seen as isomorphic, i.e. which are equal up to the laws of structural congruence given in Definition 2.

We will introduce new operators, such as the duplicator  $\nabla$  and the coduplicator  $\Delta$  (splitting respectively merging nodes), the permutation  $\rho$ , the discharger  $!$  and the codischarger  $?$  (hiding respectively creating nodes), which will be defined below (see also Figure 3). For the representation of rewriting steps as tiles, it is convenient to be able to describe these unary operators as graphs as well. In order to achieve this, we introduce an interface consisting of two sequences of nodes: root and variable nodes. Additionally we have two binary operators: composition  $;$  and a monoidal operation  $\otimes$ .

We will now describe graphs as arrows of a P-monoidal (or Part-monoidal) category [3], which can be obtained from dgs-monoidal categories [10] by adding an axiom.

P-monoidal categories are an extension of gs-monoidal categories. These describe term graphs, i.e. terms minus copying and garbage collection. Intuitively P-monoidal categories do not only contain term graphs, but also term graphs turned “upside down” and all possible combinations of these graphs.

We first give a formal definition in terms of category theory and then informally describe the meaning of the constants and operations in our setting.

**Definition 6 (P-monoidal category).** *A gs-monoidal category  $\mathbf{G}$  is a six-tuple  $(\mathbf{C}, \otimes, e, \rho, \nabla, !)$  where  $(\mathbf{C}, \otimes, e, \rho)$  is a symmetric strict monoidal category and  $! : Id \Rightarrow e : \mathbf{C} \rightarrow \mathbf{C}$ ,  $\nabla : Id \Rightarrow \otimes \circ D : \mathbf{C} \rightarrow \mathbf{C}$  are two transformations ( $D$  is the diagonal functor), such that  $!_e = \nabla_e = id_e$  and the following coherence axioms*

$$\nabla_a; id_a \otimes \nabla_a = \nabla_a; \nabla_a \otimes id_a \quad \nabla_a; id_a \otimes !_a = id_a \quad \nabla_a; \rho_{a,a} = \nabla_a$$

and the monoidality axioms

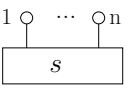
$$\nabla_{a \otimes b}; id_a \otimes \rho_{b,a} \otimes id_b = \nabla_a \otimes \nabla_b \quad !_a \otimes !_b = !_a \otimes b$$

are satisfied.

A P-monoidal category  $\mathbf{D}$  is an eight-tuple  $(\mathbf{C}, \otimes, e, \rho, \nabla, !, \Delta, ?)$  such that both the six-tuples  $(\mathbf{C}, \otimes, e, \rho, \nabla, !)$  and  $(\mathbf{C}^{op}, \otimes, e, \rho, \Delta, ?)$  are gs-monoidal categories (where  $\mathbf{C}^{op}$  is the dual category of  $\mathbf{C}$ ) and satisfy

$$\Delta_a; \nabla_a = id_a \otimes \nabla_a; \Delta_a \otimes id_a \quad \nabla_a; \Delta_a = id_a \quad ?_a; !_a = id_e$$



$id_{\underline{1}}$	$\rho_{\underline{1},\underline{1}}$	$\nabla_{\underline{1}}$	$\Delta_{\underline{1}}$	$!_{\underline{1}}$	$?_{\underline{1}}$	edge $s : \underline{n} \rightarrow \underline{0}$
$\begin{array}{c} 1 \\ \circ \\ [1] \end{array}$	$\begin{array}{cc} 1 & 2 \\ \circ & \circ \\ [2] & [1] \end{array}$	$\begin{array}{c} 1 \\ \circ \\ [1] [2] \end{array}$	$\begin{array}{cc} 1 & 2 \\ \circ & \circ \\ [1] & \end{array}$	$\begin{array}{c} 1 \\ \circ \end{array}$	$\begin{array}{c} \circ \\ [1] \end{array}$	

**Fig. 3.** P-monoidal operators

In order to model graphs we use a P-monoidal category where the objects are of the form  $\underline{n}$ ,  $n \in \mathbb{N}$ ,  $e = \underline{0}$  and  $\underline{n} \otimes \underline{m}$  is defined as  $\underline{n+m}$ . If  $\Sigma$  is a set of symbols each associated with a sort  $\underline{n} \rightarrow \underline{0}$ , then  $\mathbf{PMon}(\Sigma)$  is the P-monoidal category freely generated from the symbols in  $\Sigma$  which are interpreted as arrows.

In order to save brackets we adopt the convention that the monoidal operation  $\otimes$  takes precedence over  $;$  (the composition operator of the category). Note that by omitting the last axiom  $?_a; !_a = id_e$  we obtain exactly the definition of a dgs-monoidal category.

We depict an arrow  $t : \underline{n} \rightarrow \underline{m}$  of  $\mathbf{PMon}(\Sigma)$  by drawing a hypergraph with two sequences of external nodes:  $n$  root nodes and  $m$  variable nodes (see Figure 3). Root nodes are indicated by labels  $1, 2, \dots$ , variable nodes by labels  $[1], [2], \dots$ . The composition operator  $;$  merges the variable nodes of its first argument with the root nodes of its second argument. The tensor product  $\otimes$  takes the disjoint union of two graphs and concatenates the sequences of root respectively variable nodes of its two arguments. Note that the axiom  $?_a; !_a = id_e$  has the intuitive meaning that isolated nodes are garbage-collected.

Similar to the case of syntactic judgements it can be shown that two terms of  $\mathbf{PMon}(\Sigma)$  are equal if and only if the underlying hypergraphs are isomorphic (up to isolated nodes) [3].

There is a one-to-one correspondence between P-monoidal terms  $w : \underline{m} \rightarrow \underline{n} \in \mathbf{PMon}(\emptyset)$  (corresponding to the set of all discrete graphs) and equivalence relations on the union of  $\{r\} \times \{1, \dots, m\}$  and  $\{v\} \times \{1, \dots, n\}$ . We say that  $(r, i) \equiv_w (r, j)$  whenever the  $i$ -th and the  $j$ -th root node of  $w$  are equal, additionally  $(r, i) \equiv_w (v, j)$  whenever the  $i$ -th root node and the  $j$ -th variable node are equal and  $(v, i) \equiv_w (v, j)$  whenever the  $i$ -th and the  $j$ -th variable node are equal. An equivalence relation on a set can also be seen as a partition of this set, which is the origin of the name P(art)-monoidal category.

Syntactic judgements can be encoded into P-monoidal terms. We introduce a mapping  $\alpha$  assigning to each name its position in the sequence of external nodes. One name may appear in several positions.

**Definition 7 (Encoding of Syntactic Judgements).** Let  $\Gamma \vdash G$  be a syntactic judgement and let  $\alpha : \{1, \dots, n\} \rightarrow \Gamma$  be a surjective (but not necessarily injective) function, indicating which positions a name should occupy in the interface. We will also call  $\alpha$  an  $n$ -ary interface mapping.

Then  $[[\Gamma \vdash G]]_\alpha : \underline{n} \rightarrow \underline{0}$  is an arrow of  $\mathbf{PMon}(\Sigma)$  where  $\Sigma$  contains  $s : \underline{m} \rightarrow \underline{0}$  for every edge of the form  $s(x_1, \dots, x_m)$ . The encoding is defined as follows:

$$\begin{aligned} [[\Gamma \vdash G_1 | G_2]]_\alpha &= \nabla_{\underline{n}}; [[\Gamma \vdash G_1]]_\alpha \otimes [[\Gamma \vdash G_2]]_\alpha \\ [[\Gamma \vdash (\nu x)G]]_\alpha &= id_{\underline{n}} \otimes ?_{\underline{1}}; [[\Gamma, x \vdash G]]_{\alpha \cup \{n+1 \mapsto x\}} \text{ if } x \notin \Gamma \\ [[\Gamma \vdash nil]]_\alpha &= !_{\underline{n}} \\ [[\Gamma \vdash s(x_1, \dots, x_m)]]_\alpha &= w; s \end{aligned}$$

where  $w : \underline{n} \rightarrow \underline{m} \in \mathbf{PMon}(\emptyset)$  (the “wiring”) such that  $\equiv_w$  is the smallest equivalence containing  $\{((r, i), (v, j)) \mid \alpha(i) = x_j\}$ .

Note that if  $\alpha$  is injective, all P-monoidal terms of the form  $[[\Gamma \vdash G]]_\alpha$  lie in a subcategory of  $\mathbf{PMon}(\Sigma)$  which is generated by all symbols and constants apart from  $\Delta_{\underline{n}}$ , which means in practice that all root nodes in the interface of a graph are distinct.

*Example 5.* By encoding the syntactic judgement  $\Gamma \vdash G = y \vdash (\nu x)(\nu z)(P(x) \mid S(x, y, z) \mid P(z))$  of Example 1 with the mapping  $\alpha : \{1\} \rightarrow \{y\}$ ,  $\alpha(1) = y$  we obtain the following P-monoidal term

$$id_{\underline{1}} \otimes ?_{\underline{1}}; id_{\underline{2}} \otimes ?_{\underline{1}}; \nabla_{\underline{3}}; (!_{\underline{1}} \otimes id_{\underline{1}} \otimes !_{\underline{1}}; P) \otimes (\nabla_{\underline{3}}; (\rho_{\underline{1}, \underline{1}} \otimes id_{\underline{1}}; S) \otimes (!_{\underline{2}} \otimes id_{\underline{1}}; P)).$$

**Proposition 1.** *It holds that  $\Gamma \vdash G \equiv \Gamma' \vdash G'$  if and only if there exist injective  $\alpha, \alpha'$  such that  $[[\Gamma \vdash G]]_\alpha = [[\Gamma' \vdash G']]_{\alpha'}$ .*

## 4 A Tile Logic Representation for Synchronized Graph Rewriting with Mobility

We now describe graph rewriting in the framework of tile logic, in which we consider rewrites of the form  $s \xrightarrow[b]{a} t$  where  $s$  and  $t$  are configurations (i.e. hypergraphs) of a system and both may have root and variable nodes, their interface to the environment. The observation  $a$  describes the actions of  $s$  with respect to its root nodes, while  $b$  describes the interaction with respect to its variable nodes. The rules of tile logic describe how to derive partial rewrites and how to extend them whenever configurations are contextualized or instantiated, or—in this case—whenever two graphs are combined.

We first define the notion of a tile.

**Definition 8.** *Let  $\mathcal{H}$  and  $\mathcal{V}$  be two categories which coincide in their set of objects, which is  $\{\underline{n} \mid n \in \mathbb{N}\}$ . We call  $\mathcal{H}$  the horizontal category and  $\mathcal{V}$  the vertical category. The arrows of  $\mathcal{H}$  are also called configurations and the arrows of  $\mathcal{V}$  are called observations.*

*A tile (compare [11]) is of the form  $s \xrightarrow[b]{a} t$  where  $s : \underline{n} \rightarrow \underline{m}, t : \underline{n}' \rightarrow \underline{m}'$  are elements of  $\mathcal{H}$ , and  $a : \underline{n} \rightarrow \underline{n}', b : \underline{m} \rightarrow \underline{m}'$  are elements of  $\mathcal{V}$ .*

*Tiles can be depicted as squares (see the leftmost square in Figure 4).*

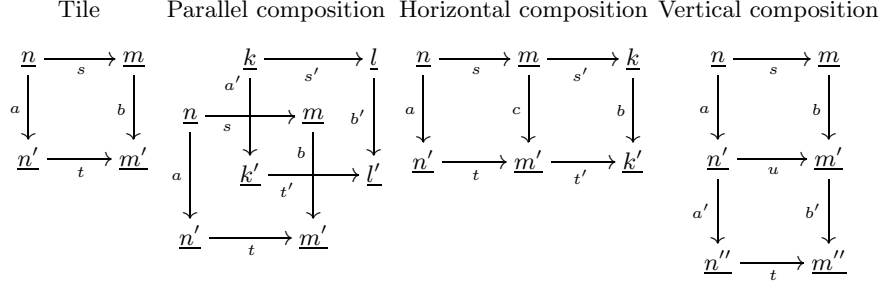


Fig. 4. Composing tiles

We can now define the more specific tiles of tile graph rewriting and the way in which they can be composed.

**Definition 9 (Tile graph rewriting).**

Let  $\Sigma = \{s : \underline{n} \rightarrow \underline{0} \mid s \text{ is an edge of arity } n\}$  and let  $\mathbf{PMon}(\Sigma)$  be the horizontal category  $\mathcal{H}$  whereas the vertical category  $\mathcal{V}$  is the free monoidal category<sup>4</sup> generated by the arrows  $a : \underline{1} \rightarrow \underline{1+n}$  for every  $a \in \text{Act} \setminus \{\varepsilon\}$  with  $\text{ar}(a) = n$ . The idle action  $\varepsilon$  corresponds to the identity arrow  $\text{id}_{\underline{1}}$ .

Tiles can be constructed in the following way: a tile is either taken from a fixed set  $\mathcal{R}$  of generator tiles, or it is a reflexive tile (*h-refl*) or (*v-refl*), or it is one of the auxiliary tiles (*dupl*), (*codupl*), (*disch*), (*codisch*) or (*perm*), or it is obtained by parallel composition (*p-comp*), horizontal composition (*h-comp*) or vertical composition (*v-comp*) (see also Figure 4).

We write  $\mathcal{R} \Vdash s \xrightarrow[a]{a} t$  whenever this tile can be derived from the generator tiles in  $\mathcal{R}$ .

$$\begin{array}{lll}
(h\text{-refl}) \frac{s : \underline{n} \rightarrow \underline{m} \in \mathcal{H}}{s \xrightarrow[id_m]{id_n} s} & (v\text{-refl}) \frac{a : \underline{n} \rightarrow \underline{m} \in \mathcal{V}}{id_{\underline{n}} \xrightarrow[a]{a} id_{\underline{m}}} & (dupl) \frac{a : \underline{n} \rightarrow \underline{m} \in \mathcal{V}}{\nabla_{\underline{n}} \xrightarrow[a \otimes a]{a} \nabla_{\underline{m}}} \\
(codupl) \frac{a : \underline{n} \rightarrow \underline{m} \in \mathcal{V}}{\Delta_{\underline{n}} \xrightarrow[a]{a \otimes a} \Delta_{\underline{m}}} & (disch) \frac{a : \underline{n} \rightarrow \underline{m} \in \mathcal{V}}{!_{\underline{n}} \xrightarrow[id_0]{a} !_{\underline{m}}} & (codisch) \frac{a : \underline{n} \rightarrow \underline{m} \in \mathcal{V}}{?_{\underline{n}} \xrightarrow[a]{id_0} ?_{\underline{m}}}
\end{array}$$

<sup>4</sup> Given a set  $A$  of arrows, the free monoidal category generated by  $A$  consists of all arrows which can be obtained from composing the arrows of  $A$  with the composition operator  $;$  and the monoidal operator  $\otimes$ , observing the category axioms ( $;$  is associative and  $\varepsilon = id_{\underline{1}}$  is its unit), the monoidality axioms ( $\otimes$  is associative and  $id_{\underline{0}}$  is its unit) and  $a_1; a'_1 \otimes a_2; a'_2 = (a_1 \otimes a_2); (a'_1 \otimes a'_2)$ .

$$\begin{array}{c}
\text{(perm)} \frac{a : \underline{n} \rightarrow \underline{m}, b : \underline{n}' \rightarrow \underline{m}' \in \mathcal{V}}{\rho_{\underline{n}, \underline{n}'} \xrightarrow{a \otimes b} \rho_{\underline{m}, \underline{m}'}} \\
\text{(h-comp)} \frac{s \xrightarrow{a} t, s' \xrightarrow{c} t'}{s; s' \xrightarrow{a} t; t'} \\
\text{(p-comp)} \frac{s \xrightarrow{a} t, s' \xrightarrow{a'} t'}{s \otimes s' \xrightarrow{a \otimes a'} t \otimes t'} \\
\text{(v-comp)} \frac{s \xrightarrow{a} u, u \xrightarrow{a'} t}{s \xrightarrow{a; a'} t}
\end{array}$$

We first show that if the generator tiles exhibit a certain well-formedness property, then we can construct every tile in the following way: first, we can use all rules apart from  $(v\text{-comp})$  in order to construct several tiles which, finally, can be combined with rule  $(v\text{-comp})$ . This says, basically, that it is sufficient to examine tiles which describe one single rewriting step.

**Proposition 2.** *We assume that the set  $\mathcal{R}$  of generator tiles satisfies the following properties: let  $s \xrightarrow{a} t$  be a generator tile, then it holds that  $s \in \Sigma$  and furthermore there are actions  $a_1, \dots, a_n \in \text{Act} \setminus \{\varepsilon\}$ , such that  $a = a_1 \otimes \dots \otimes a_n$  and  $b = id_0$ .*

*Now let  $\mathcal{R} \Vdash s \xrightarrow{a} t$ . Then it holds that there are configurations  $s = s_0, s_1, \dots, s_m = t$  and observations  $a'_1, \dots, a'_m, b'_1, \dots, b'_m$  such that  $\mathcal{R} \Vdash s_{i-1} \xrightarrow{a'_i} s_i$  for  $i \in \{1, \dots, n\}$  and the respective tiles can be derived without rule  $(v\text{-comp})$ . Furthermore  $a = a'_1; \dots; a'_m$  and  $b = b'_1; \dots; b'_m$ .*

We can now formulate one of the two main results of this paper: the operational correspondence between rewriting of syntactic judgements and of P-monoidal terms.

We first introduce the following notation: let  $x_1 \dots x_n$  be a string of names. By  $\alpha = \langle x_1 \dots x_n \rangle$  we denote the interface mapping  $\alpha : \{1, \dots, n\} \rightarrow \{x_1, \dots, x_n\}$  where  $\alpha(i) = x_i$ .

**Proposition 3 (Operational Correspondence).** *Let  $R$  be a set of rewriting rules on syntactic judgements. We define a set  $\mathcal{R}$  of generator tiles as follows:*

$$\mathcal{R} = \left\{ s \xrightarrow{id_0^{a_1 \otimes \dots \otimes a_m}} \llbracket \Gamma' \vdash G' \rrbracket_{\langle x_1 \tilde{y}_1 \dots x_m \tilde{y}_m \rangle} \mid (x_1, \dots, x_m \vdash s(x_1, \dots, x_m)) \xrightarrow{\Lambda} \Gamma' \vdash G' \in R, a_i = \text{act}_\Lambda(x_i), \tilde{y}_i = n_\Lambda(x_i) \right\}.$$

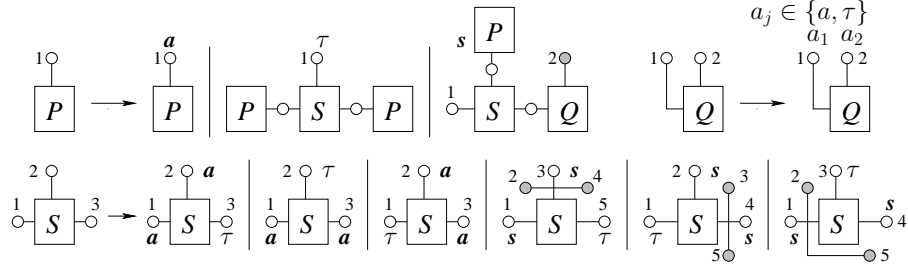
– *It holds that  $R \Vdash (\Gamma \vdash G \xrightarrow{\Lambda} \Gamma' \vdash G')$  implies*

$$\mathcal{R} \Vdash (\llbracket \Gamma \vdash G \rrbracket_\alpha \xrightarrow{id_0^{a_1 \otimes \dots \otimes a_m}} \llbracket \Gamma' \vdash G' \rrbracket_{\langle \alpha(1) \tilde{y}_1 \dots \alpha(m) \tilde{y}_m \rangle}) \text{ where } a_i = \text{act}_\Lambda(\alpha(i)), \tilde{y}_i = n_\Lambda(\alpha(i)).$$

- And it holds that if  $\mathcal{R} \Vdash (\llbracket \Gamma \vdash G \rrbracket_\alpha \xrightarrow[id_\Omega]{a_1 \otimes \dots \otimes a_m} t)$  for some  $P$ -monoidal term  $t$ , then  $R \Vdash (\Gamma \vdash G \xrightarrow{\Lambda} \Gamma' \vdash G')$  where  $a_i = \text{act}_\Lambda(\alpha(i))$ ,  $\tilde{y}_i = n_\Lambda(\alpha(i))$  and  $\llbracket \Gamma' \vdash G' \rrbracket_{\langle \alpha(1)\tilde{y}_1 \dots \alpha(m)\tilde{y}_m \rangle} = t$ .

*Proof (Sketch).* The first half of the proposition can be shown by induction on the inference rules applied. The second half of the proposition is shown by induction on the syntactic structure of  $\Gamma \vdash G$  and with the decomposition property (Proposition 4 which will be proved in Section 5 without referring back to this proposition).  $\square$

*Example 6.* Encoding the rewrite rules on syntactic judgements from Example 2 into generator tiles gives us the tiles depicted in Figure 5.



**Fig. 5.** Generator tiles

Here we represent a tile of the form  $s \xrightarrow[id_\Omega]{a_1 \otimes \dots \otimes a_n} t$  by drawing  $s$  and  $t$  as graphs and by labelling the free nodes of  $t$  by the actions  $a_1, \dots, a_n$ . Specifically if  $N_i = \sum_{j=1}^i (ar(a_j) + 1)$ , then the  $N_{i-1} + 1$ -st free node of  $t$  is labelled  $a_i$ , while the next  $ar(a_i) - 1$  nodes stay unlabelled (and are shaded grey in the graphical representation), indicating that these nodes are generated by the action  $a_i$ . Two nodes that are connected by a line represent one and the same node.

## 5 Bisimilarity is a Congruence

Based on tiles we can now define the notions of bisimulation and bisimilarity and thus define a notion of an observable, compositional equivalence on graphs.

**Definition 10 (Bisimulation on tiles).** *Given a labelled transition system, a bisimulation is a symmetric, reflexive relation  $\sim$  on the states of the transition system, such that if  $s \sim t$  and  $s \xrightarrow{a} s'$ , then there exists a transition  $t \xrightarrow{a} t'$  such that  $s' \sim t'$ . We say that two states  $s$  and  $t$  are bisimilar ( $s \simeq t$ ) whenever there is a bisimulation  $\sim$  such that  $s \sim t$ .*

In tile graph rewriting the tile  $s \xrightarrow[b]{a} t$  is considered to be a transition with label  $(a, b)$ . We say that two configurations  $s, t$  are bisimilar wrt. a set  $\mathcal{R}$  of generator tiles (in symbols  $s \simeq_{\mathcal{R}} t$ ) whenever  $s$  and  $t$  are bisimilar in the transition system generated by  $\mathcal{R}$ .

It is already known that bisimilarity is a congruence whenever the underlying tile system satisfies the following decomposition property.

**Definition 11 (Decomposition Property).** *A tile system satisfies the decomposition property if for all tiles  $s \xrightarrow[b]{a} t$  entailed by the tile system, it holds that (1) if  $s = s_1; s_2$  then there exist  $c \in \mathcal{V}, t_1, t_2 \in \mathcal{H}$  such that  $s_1 \xrightarrow[c]{a} t_1, t_1 \xrightarrow[b]{c} t_2$  and  $t = t_1; t_2$  (2) if  $s = s_1 \otimes s_2$  then there exist  $a_1, a_2, b_1, b_2 \in \mathcal{V}, t_1, t_2 \in \mathcal{H}$  such that  $s_1 \xrightarrow[b_1]{a_1} t_1, s_2 \xrightarrow[b_2]{a_2} t_2, a = a_1 \otimes a_2, b = b_1 \otimes b_2$  and  $t = t_1 \otimes t_2$ .*

**Proposition 4 (cf. [11]).** *If a tile system satisfies the decomposition property, then bisimilarity defined on its transition system is a congruence.*

Similar to the case of de Simone [5] or `tyft/tyxt`-formats [12], there is a sufficient syntactical property ensuring that bisimilarity is indeed a congruence, which is stated in the second main result of this paper.

**Proposition 5.** *If, in tile graph rewriting, all generator tiles satisfy the basic source property, i.e. if for every generator tile  $s \xrightarrow[b]{a} t$  it holds that  $s \in \Sigma$ , then the decomposition property holds. Thus bisimilarity is a congruence.*

*Proof (Sketch).* By induction on the derivation of a tile, following the lines of a similar proof in [2].

**Corollary 1.** *All the tile graph rewriting systems derived from rewriting rules on syntactic judgements satisfy the basic source property. Thus the decomposition property holds and bisimilarity is a congruence.*

Having established that bisimilarity is indeed a congruence in the case of tile graph rewriting we now transfer this result back to rewriting on syntactic judgements with the help of the operational correspondence (Proposition 3). First we have to define bisimulation on syntactic judgements. We use the following intuition: an observer from the outside has access to the external nodes of a graph, however he or she is not able to determine their names and he or she should also not be able to find out whether or not two nodes are equal. So, given two syntactic judgements, we add an interface mapping  $\alpha$  which assigns numbers to names and in this way hides the internal details from an external observer.

For the next definition remember that the mapping  $i \mapsto x_i$  is denoted by  $\langle x_1 \dots x_n \rangle$ .

**Definition 12 (Bisimulation on syntactic judgements).** Let  $\Gamma \vdash G$  be a syntactic judgement. An  $n$ -ary interface for a syntactic judgement is a surjective mapping  $\alpha : \{1, \dots, n\} \rightarrow \Gamma$ , as defined in Definition 7.

A symmetric, reflexive relation  $\sim$  on pairs consisting of syntactic judgements and their corresponding interfaces is called a bisimulation (wrt. a set  $R$  of rewriting rules) if whenever  $(\Gamma_1 \vdash G_1, \alpha_1) \sim (\Gamma_2 \vdash G_2, \alpha_2)$ , then

- there is an  $n \in \mathbb{N}$  such that  $\alpha_1$  and  $\alpha_2$  are both  $n$ -ary interfaces
- whenever  $\Gamma_1 \vdash G_1 \xrightarrow{A_1} \Gamma'_1 \vdash G'_1$  with  $\Lambda_1(\alpha_1(i)) = (a_i, \tilde{y}_i)$ , then it holds that  $\Gamma_2 \vdash G_2 \xrightarrow{A_2} \Gamma'_2 \vdash G'_2$  with  $\Lambda_2(\alpha_2(i)) = (a_i, \tilde{z}_i)$  and

$$(\Gamma'_1 \vdash G'_1, \langle \alpha_1(1)\tilde{y}_1 \dots \alpha_1(n)\tilde{y}_n \rangle) \sim (\Gamma'_2 \vdash G'_2, \langle \alpha_2(1)\tilde{z}_1 \dots \alpha_2(n)\tilde{z}_n \rangle).$$

We say that two pairs  $(\Gamma_1 \vdash G_1, \alpha_1)$  and  $(\Gamma_2 \vdash G_2, \alpha_2)$  are bisimilar (wrt. a set  $R$  of rewriting rules) whenever there is a bisimulation  $\sim$  (wrt. a set  $R$  of rewriting rules) such that  $(\Gamma_1 \vdash G_1, \alpha_1) \sim (\Gamma_2 \vdash G_2, \alpha_2)$ . Bisimilarity on syntactic judgements is denoted by the symbol  $\simeq_R$ .

In order to show that bisimilarity on syntactic judgements is a congruence with respect to parallel composition and hiding, we need the following result on full abstraction.

**Proposition 6 (Full abstraction).** The encoding  $\llbracket \_ \rrbracket_\alpha$  is fully abstract in the following sense: for any set  $R$  of rewriting rules it holds that

$$(\Gamma_1 \vdash G_1, \alpha_1) \simeq_R (\Gamma_2 \vdash G_2, \alpha_2) \iff \llbracket \Gamma_1 \vdash G_1 \rrbracket_{\alpha_1} \simeq_{\mathcal{R}} \llbracket \Gamma_2 \vdash G_2 \rrbracket_{\alpha_2}$$

where  $\mathcal{R}$  is defined as in Proposition 3.

*Proof (Sketch).* Straightforward by regarding the respective definitions of bisimilarity, from Proposition 2 and the operational correspondence result in Proposition 3.  $\square$

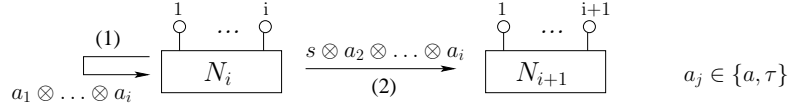
Now it is straightforward to show that bisimilarity on syntactic judgements is a congruence as well.

**Proposition 7.** Let  $R$  be a set of rewriting rules and let  $\mathcal{R}$  be the corresponding set of generator tiles defined as in Proposition 3.

We assume that  $(\Gamma_1, X_1 \vdash G_1, \alpha_1) \simeq_R (\Gamma_2, X_2 \vdash G_2, \alpha_2)$  such that  $\alpha_i : \{1, \dots, n+m\} \rightarrow \Gamma_i \cup X_i$  and  $\alpha_i^{-1}(X_i) = \{n+1, \dots, n+m\}$  for  $i \in \{1, 2\}$ . Then it holds that  $(\Gamma_1 \vdash (\nu X_1)G_1, \alpha_1|_{\{1, \dots, n\}}) \simeq_R (\Gamma_2 \vdash (\nu X_2)G_2, \alpha_2|_{\{1, \dots, n\}})$ .

And if  $(\Gamma_1 \vdash G_1, \alpha_1) \simeq_R (\Gamma_2 \vdash G_2, \alpha_2)$  and  $(\Gamma_1 \vdash G'_1, \alpha_1) \simeq_R (\Gamma_2 \vdash G'_2, \alpha_2)$ , then it follows that  $(\Gamma_1 \vdash G_1 \mid G'_1, \alpha_1) \simeq_R (\Gamma_2 \vdash G_2 \mid G'_2, \alpha_2)$ .

*Proof (Sketch).* Straightforward by using the full abstraction result from Proposition 6, by using that fact that bisimilarity on P-monoidal terms is a congruence (see Proposition 5) and by regarding the definition of the encoding  $\llbracket \_ \rrbracket_\alpha$  in Definition 7.  $\square$



**Fig. 6.** Specification of the communication network

## 6 Bisimulation up-to Congruence

In order to show that two graphs are bisimilar in practice, we need a proof technique for bisimulation, a so-called bisimulation up-to congruence (for up-to techniques see also [18]).

**Definition 13.** For a given relation  $B$  on  $P$ -monoidal terms, we denote by  $\equiv_B$  the smallest congruence (with respect to the operators  $;$  and  $\otimes$ ) that contains  $B$ .

A symmetric relation  $B$  is called a bisimulation up-to congruence whenever  $(s, t) \in B$  and  $s \xrightarrow[a]{a} s'$  imply  $t \xrightarrow[a]{a} t'$  and  $s' \equiv_B t'$ .

**Proposition 8.** If the decomposition property holds for the respective tile logic and  $B$  is a bisimulation up-to congruence, then  $\equiv_B$  is a bisimulation.

It is typically easier to show that  $B$  is a bisimulation up-to congruence than to show that  $\equiv_B$  is a bisimulation, mainly because  $B$  can be much smaller than  $\equiv_B$  and so there are fewer cases to consider. It may even be the case that  $B$  is finite and  $\equiv_B$  is infinite in size.

## 7 Example: Communication Network

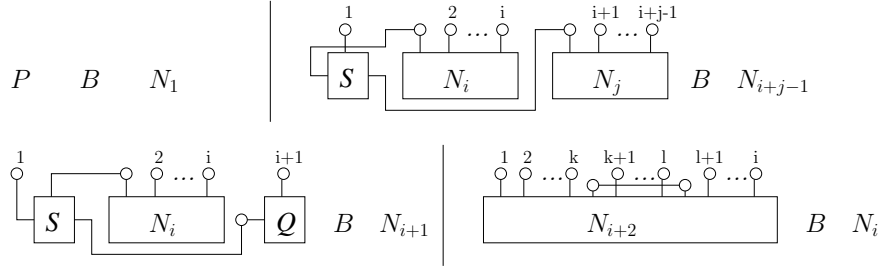
We return to our running example and intend to investigate which steps a single process can perform, or rather which are the actions that are observable from the outside. To this aim we give a specification  $N_1$  which models in a single edge the entire communication topology  $P$  may generate. Note that a process  $P$  may start with a single free node, but can create new free nodes by performing  $s$  actions. The specification has to take this into account and  $N_1$  may therefore reduce to  $N_2, N_3$  etc., where  $N_i : \underline{i} \rightarrow \underline{0}$ .

The generator tiles for the specification are depicted in Figure 6, where this time we put the observations on the arrows rather than on the free nodes of the right-hand side.

The edge  $N_i$  can either perform an arbitrary combination of  $a$  and  $\tau$  actions and stay  $N_i$  or it can perform an  $s$  action on its first node and  $a$ 's and  $\tau$ 's on the remaining nodes and become  $N_{i+1}$ .

In order to show that  $P$  and  $N_1$  are indeed bisimilar we proceed as follows: We consider the tile system generated by both the tiles belonging to processes and switches and the tiles of the specification, and denote the combined set of generator tiles by  $\mathcal{R}$ . Since the set of edges involved in the first set of generator





**Fig. 7.** Example of a bisimulation up-to congruence

tiles is disjoint from that in the second set, the rules can not interfere with each other and  $P$  respectively  $N_1$  can not perform more reductions with the additional tiles.

**Proposition 9.** *The symmetric closure of the relation  $B$  depicted in Figure 7 is a bisimulation up-to congruence. Since the basic source property and therefore the decomposition property hold, it follows that  $\equiv_B$  is a bisimulation.*

*And since  $(P, N_1) \in B$ , we conclude that  $P \simeq_{\mathcal{R}} N_1$ .*

*Proof (Sketch).* From Proposition 2 it follows that it is sufficient to regard only tiles which can be composed without using rule (*v-comp*).

We exemplarily treat the following case: let  $(P, N_1) \in B$  and we assume that  $P$  performs a  $\tau$  action and replaces itself with a switch and two processes, i.e. the second rewrite rule for  $P$  is applied. In this case  $N_1 \xrightarrow[id_0]{\tau} N_1$  and we have to show that the two resulting graphs are in the  $\equiv_B$ -relation:

$$\begin{array}{c} \textcircled{1} \\ | \\ \boxed{P} \end{array} \xrightarrow{\tau} \begin{array}{c} \textcircled{1} \\ | \\ \boxed{P} \end{array} \text{---} \begin{array}{c} \textcircled{1} \\ | \\ \boxed{S} \end{array} \text{---} \begin{array}{c} \textcircled{1} \\ | \\ \boxed{P} \end{array} \equiv_B \begin{array}{c} \textcircled{1} \\ | \\ \boxed{N_1} \end{array} \text{---} \begin{array}{c} \textcircled{1} \\ | \\ \boxed{S} \end{array} \text{---} \begin{array}{c} \textcircled{1} \\ | \\ \boxed{N_1} \end{array} \equiv_B \begin{array}{c} \textcircled{1} \\ | \\ \boxed{N_1} \end{array} \xleftarrow{\tau} \begin{array}{c} \textcircled{1} \\ | \\ \boxed{N_1} \end{array}$$

□

The scenario we have presented resembles the view of a user which starts with one single port of access to a network which may be huge. The user can request further connections into the network (with an  $s$  action) and he or she can send signals  $a$  which are received in the network. However, in whatever way the user interacts with the entire network, its topology will always be hidden, its expansion unobservable and it thus constitutes a black box. Internal communications, of which several may take place in parallel, are indistinguishable from  $\tau$ -steps and thus completely hidden from the environment.

If, however, we start with a disconnected graph with  $i$  external nodes and compare it to an edge  $N_i$ , the two expressions are *not* bisimilar. Consider for example the two graphs  $P \otimes P$  and  $N_2$ , both of arity 2. We observe that  $P \otimes P \xrightarrow[id_0]{a \otimes id_1} P \otimes P$ , whereas  $N_2$  can not match this transition. If we assume that

the first node of  $N_2$  produces an action  $a$ , we either get  $a$  or  $\tau$  as the action of the second node, but we never get  $id_{\underline{1}}$ . In general we can state that whenever we have a graph  $t$  consisting of processes and switches, we can determine its connected external nodes in the following way: a set of external nodes is connected if and only if there is a transition such that exactly the nodes of the set perform an action different from  $id_{\underline{1}} = \varepsilon$ , and that furthermore there is no proper subset with the same property.

Another scenario would be to start with a graph with several external nodes of which two or more are connected via switches. In this case an  $s$  action originating on one of the external nodes may be routed to a different external node, giving us two new nodes in the interface, which, however, must be equal.

## 8 Conclusion

We have presented synchronized graph rewriting with mobility for two forms of graph representation (syntactic judgements and arrows in a P-monoidal category) and we have shown that bisimilarity for synchronized graph rewriting is a congruence with respect to graph composition. A tile logic semantics for synchronized graph rewriting without mobility has already been defined in [23], whereas synchronized graph rewriting with mobility has so far only been considered for syntactic judgements [13, 14], but not in the context of tile logics. Moreover no equivalence of graphs based on observations has been introduced there.

In [14] not only Hoare synchronization, but also Milner synchronization is treated and an encoding of the  $\pi$ -calculus into synchronized graph rewriting is given, using Milner synchronization.

An earlier form of synchronized graph rewriting has been treated in [6]. Furthermore, the mobility treated in this paper is reminiscent of the rendezvous mechanism presented in [4].

In general we know of little work concerning the definition of observational equivalences for graph rewriting. As already mentioned in the introduction there are basically two ways to go when one wants to introduce bisimilarity on graphs. The first alternative would be to base the theory on unlabelled production as in the double-pushout approach [7]. Without labels on the transitions it is necessary to define canonical forms of graphs, in order to be able to observe something. Work by Fernández and Mackie on interaction nets [9] and by Yoshida on process graphs [25] goes in that direction.

In  $\pi$ -calculus, for example, the canonical forms mentioned above are called “barbs” and a process has a barb for channel  $c$  whenever it is able to perform an input or output on  $c$ . The resulting bisimulation is therefore called barbed bisimulation [22], which ordinarily does not induce a congruence, and the definition of the smallest congruence containing barbed bisimilarity requires universal quantification over all possible contexts. This, however, makes actual proofs of bisimilarity complicated.

In this paper, however, we chose to use synchronized graph rewriting as a framework. We model transitions whose transition labels (observations) describe

exactly the interaction of a single edge with its environment. This enables us to define a simple syntactic property on the rewriting rules which ensures that bisimilarity is a congruence. Existing work is mainly related to action calculi [20, 17] which also have a graphical representation.

As we have seen in the example in Section 7, the bisimilarity defined in this paper is rather coarse-grained: it can determine whether a network is connected or disconnected, but we can, for example, not determine the degree of parallelism in a network. In order to be able to do this, it would be necessary to establish a concurrent semantics for synchronized graph rewriting. Another interesting extension would be to enrich the notion of observation: so far we are only able to observe the actions performed on external nodes, but we are not able to determine whether, for example, two nodes are connected by an edge. It seems therefore promising to extend the framework in such a way that we are allowed to observe occurrences of specific subgraphs.

**Acknowledgements:** We would like to thank Roberto Bruni and Dan Hirsch for their help.

## References

1. R. Bruni, D. Frutos-Escrig, N. Martí-Oliet, and U. Montanari. Bisimilarity congruences for open terms and term graphs via tile logic. In *Proc. of CONCUR 2000*, pages 259–274. Springer-Verlag, 2000. LNCS 1877.
2. R. Bruni, D. Frutos-Escrig, N. Martí-Oliet, and U. Montanari. Tile bisimilarity congruences for open terms and term graphs. Technical Report TR-00-06, Dipartimento di Informatica, Università di Pisa, 2000.
3. Roberto Bruni, Fabio Gadducci, and Ugo Montanari. Normal forms for partitions and relations. In *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT '98*, pages 31–47. Springer-Verlag, 1998. LNCS 1589, full version to appear in TCS.
4. Gnanamalar David, Frank Drewes, and Hans-Jörg Kreowski. Hyperedge replacement with rendezvous. In *Proc. of TAPSOFT (Theory and Practice of Software Development)*, pages 167–181. Springer-Verlag, 1993. LNCS 668.
5. Roberto de Simone. Higher level synchronizing devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.
6. P. Degano and U. Montanari. A model of distributed systems based on graph rewriting. *Journal of the ACM*, 2(34):411–449, 1987.
7. H. Ehrig. Introduction to the algebraic theory of graphs. In *Proc. 1st International Workshop on Graph Grammars*, pages 1–69. Springer-Verlag, 1979. LNCS 73.
8. H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.3: Concurrency, Parallellism, and Distribution*. World Scientific, 1999.
9. Maribel Fernández and Ian Mackie. Coinductive techniques for operational equivalence of interaction nets. In *Proc. of LICS '98*. IEEE Computer Society Press, 1998.
10. F. Gadducci and R. Heckel. An inductive view of graph transformation. In *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT '97*, pages 223–237. Springer-Verlag, 1997. LNCS 1376.

11. F. Gadducci and U. Montanari. The tile model. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1999.
12. J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1992.
13. Dan Hirsch, Paola Inverardi, and Ugo Montanari. Reconfiguration of software architecture styles with name mobility. In António Porto and Gruiua-Catalin Roman, editors, *Proc. of COORDINATION 2000*, pages 148–163. Springer-Verlag, 2000. LNCS 1906.
14. Dan Hirsch and Ugo Montanari. Synchronized hyperedge replacement with name mobility (a graphical calculus for mobile systems). In *Proc. of CONCUR '01*, pages 121–136. Springer-Verlag, 2001. LNCS 2154.
15. Barbara König. *Description and Verification of Mobile Processes with Graph Rewriting Techniques*. PhD thesis, Technische Universität München, 1999.
16. Barbara König. A graph rewriting semantics for the polyadic  $\pi$ -calculus. In *Workshop on Graph Transformation and Visual Modeling Techniques (Geneva, Switzerland), ICALP Workshops '00*, pages 451–458. Carleton Scientific, 2000.
17. James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *Proc. of CONCUR 2000*, 2000. LNCS 1877.
18. R. Milner and D. Sangiorgi. Techniques of weak bisimulation up-to. In *Proc. of CONCUR '92*. Springer-Verlag, 1992. LNCS 630.
19. Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1980. LNCS 92.
20. Robin Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, 1996.
21. Robin Milner. *Communicating and Mobile Systems: the  $\pi$ -Calculus*. Cambridge University Press, 1999.
22. Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *Proc. of ICALP '92*. Springer-Verlag, 1992. LNCS 623.
23. U. Montanari and F. Rossi. Graph rewriting, constraint solving and tiles for coordinating distributed systems. *Applied Categorical Structures*, 7:333–370, 1999.
24. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations*, volume 1. World Scientific, 1997.
25. Nobuko Yoshida. Graph notation for concurrent combinators. In *Proc. of TPPP '94*. Springer-Verlag, 1994. LNCS 907.