A Rewriting Calculus for Multigraphs with Ports

Oana Andrei, Hélène Kirchner

▶ **To cite this version:**

Oana Andrei, Hélène Kirchner. A Rewriting Calculus for Multigraphs with Ports. [Technical Report] 2007, pp.18. inria-00139363v1

HAL Id: inria-00139363

https://inria.hal.science/inria-00139363v1

Submitted on 30 Mar 2007 (v1), last revised 21 May 2007 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards a Rewriting Calculus for Multigraphs with Ports

Oana Andrei[1] and Hélène Kirchner[2]

[1] INRIA - LORIA
[2] CNRS - LORIA
Campus scientifique BP 239
F-54506 Vandoeuvre-lès-Nancy Cedex, France
First.Last@loria.fr

Abstract. In this paper, we define labeled multigraphs with ports, a graph model which makes explicit connection resources in each node, allows multiple edges and loops. Dynamic evolution of these structures is expressed with multigraph rewrite rules and a multigraph rewriting relation. Then we encode the multigraphs and multigraph rewriting using algebraic terms and term rewriting to provide a more operational semantics of the multigraph rewriting relation. This term version can be embedded in the rewriting calculus, thus providing for labeled multigraphs transformations a high-level calculus, called ρ_{mg}-calculus, with good properties and expressive power.

1 Introduction

Graphs are high-level constructs widely used for describing complex structures, like communication networks, neural networks, UML diagrams, microprocessor design, XML documents, biological systems. Graph transformation provides a rule-based modeling of their dynamic evolution. Different approaches have been proposed to formalize graph transformation and to define graph rewriting, summarized for instance in [17].

We have explored graph models for simulation of chemical reactors [7, 2] and of protein interactions [3], and we found the need in this contexts for graph structures where edges expressing connections between nodes are explicitly partitioned according to different node resources.

In this paper, we propose a graph model which makes explicit the connection resources in each node, and in this way, provides a finer grained modeling of a problem. We identify a quite general class of directed graphs where nodes are embedded with an explicit information on their resources for connections, allowing multiple edges and loops. We call these node resources *ports*. We represent node information as labels and we call such graphs *labeled multigraphs with ports*; the edge labels are ordered pairs of the two ports which determine it.

The concept of port for graphs is not a novelty. It can be seen as a refinement of the connectivity information for nodes. We were inspired in investigating this graph particularity by the binding sites of proteins [12, 6]. In a different context, the Graph Markup Language GraphML [8], an XML format for graph structures with background in the graph drawing community, uses ports in nodes for partitioning the incidences.

The paper is divided in three parts: after basic definitions in Sect. 2, we first define in Sect. 3,the labeled multigraphs with ports, multigraph rewrite rules, and the multigraph rewriting relation; second, in Sect. 4, we encode the multigraphs and multigraph rewriting using algebraic terms and term rewriting to provide a more operational semantics of the multigraph rewriting relation; and third, in Sect. 5, we embed the term approach on multigraph rewriting in the rewriting calculus obtaining for free a rewriting calculus for labeled multigraphs, called ρ_{mg}-calculus. Therefore we provide for the multigraph rewriting a high-level calculus extending algebraic rewriting and we can take benefit of the nice properties of the ρ-calculus and of the possibility of using rewriting strategies and rule conditions to control rule application. The operational correspondence result in Sect. 4 allows us to express visually quite complex multigraph transformations and perform them using term rewriting. In Sect. 6 we present some implementation hints for multigraph rewriting, as well as some extensions and applications for multigraphs with ports.

2 Background

In this section we briefly review some basic definitions of order-sorted algebra ([13]), term rewriting ([14, 4]), graph theory and graph transformation ([11, 17]) used in this paper.

Term Algebra. A *many-sorted signature* is a pair (S, Σ), where S is called the *sort set* and Σ is an $S^* \times S$-sorted family $\{\Sigma_{w,s} \mid w \in S^* \text{ and } s \in S\}$. For $\mathcal{X} = \{\mathcal{X}_s\}$ an S-sorted family of disjoint sets of variables, $\mathcal{T}_\Sigma(\mathcal{X})$ is the smallest set of *Σ-terms over $\mathcal{X}$* built with operators from Σ and variables from $\mathcal{X}$. The set of all terms of sort s is denoted by $\mathcal{T}_{\Sigma,s}(\mathcal{X})$. Positions in a term are represented as sequences of integers. The empty sequence ϵ denotes the top position. The notation $t[s]_p$ emphasizes that s is a subterm of t occurring at position p. A *substitution* is a partial mapping from $\mathcal{X}$ to terms and it uniquely extends to a Σ-endomorphism on $\mathcal{T}_\Sigma(\mathcal{X})$. A finite substitution is given as $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, with $x_i \in \mathcal{X}$ and $t_i \in \mathcal{T}_\Sigma(\mathcal{X})$, for all $i = 1..n$. Considering for S a partially ordered set leads to quite similar definitions for order-sorted signatures and order-sorted terms [13].

Term Rewriting. A set $\mathcal{R}$ of *rewrite rules* is a set of pairs of terms of $\mathcal{T}_\Sigma(\mathcal{X})$, denoted $l \to r$, such that l and r belong to the same sort, $l \notin \mathcal{X}$ and $Var(r) \subseteq Var(l)$. In this paper, we only consider finite sets of rewrite rules. The *rewriting relation* induced by $\mathcal{R}$ is denoted by $\to_\mathcal{R}$ ($\to$ if there is no ambiguity on $\mathcal{R}$), and defined by $s \to t$ iff there exists a substitution σ and a position p in s such that $s = s[\sigma l]_p$ for some rule $l \to r$ of $\mathcal{R}$, and $t = s[\sigma r]_p$. This is written $s \to_\mathcal{R}^{p,l\to r,\sigma} t$ where either $\mathcal{R}$, p, $l \to r$, or σ may be omitted. The reflexive transitive closure of the rewriting relation induced by $\mathcal{R}$ is denoted by $\overset{*}{\to}_\mathcal{R}$. A rewriting derivation is a chain of terms $t_1 \to t_1 \to \ldots t_n$. If $t \overset{*}{\to} t'$ and t' cannot be rewritten anymore then t' is called a *normal form* of t denoted by $t \downarrow_\mathcal{R}$. The existence of a unique normal form is guaranteed in particular when $\mathcal{R}$ is confluent (two derivations issued from the same term eventually converge to a common term) and strongly terminating (every derivation terminates).

A *rewrite theory* on $\mathcal{T}_\Sigma(\mathcal{X})$ is a triple $(\Sigma, E, \mathcal{R})$ where E is a finite set of (sort-preserving) equalities and $\mathcal{R}$ a finite set of rewrite rules. The relation $\to_{\mathcal{R}/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $=_E; \to_\mathcal{R}; =_E$. It induces a relation $\to_{\mathcal{R}/E}$ on the quotient algebra $\mathcal{T}_\Sigma(\mathcal{X})/_{=_E}$ by $[t]_E \to_{\mathcal{R}/E} [t']_E$ iff $t \to_{\mathcal{R}/E} t'$. The relation $\to_{\mathcal{R},E}$ (called rewriting modulo E) on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined by $s \to_{\mathcal{R},E} t$ iff there exists a substitution σ and a position p in s such that $s|_p =_E \sigma l$ for some rule $l \to r$ of $\mathcal{R}$, and $t = s[\sigma r]_p$. In this paper, we will consider theories E defined by specific axioms,

namely associativity (A), associativity and commutativity (AC) and unit element (U) of certain function symbol.

Rewriting (rewriting modulo E) involves matching problems: a matching problem $t_1 \ll t$ is successful if there exist a substitution σ such that $\sigma(t_1) = t$ (resp. $\sigma(t_1) =_E t$). Matching (rewriting) modulo associative (A), associative with unit element (AU), associative-commutative (AC), or associative-commutative with unit element (ACU) theories, is an essential operation when modeling structures like list, sets or multisets using term rewriting. Usually an associative operator is represented using a *flattened* form. For example, the term $f(f(f(s,t),u),f(s,v))$ built using an associative operator f and the subterms s,t,u,v having top operators different from f, has the (unique) flattened form $f^*(s,t,u,s,v)$ where f^* is a variadic operator. When no confusion is likely to result, we use the same symbol for an operator and its flattened form.

Labeled Graphs. A *label alphabet* $\mathcal{L} = (\mathcal{L}_N, \mathcal{L}_E)$ is a pair of sets of node labels and edge labels. A (finite) *graph* over $\mathcal{L}$ is a triple $G = (N, E, l)$ where N is a set $\{n_1, \ldots, n_k\}$ of elements called *nodes*, E is a family $(e_1, \ldots, e_m)$ of elements of the Cartesian product $N \times N$ called *edges*, and $l = (l_N, l_E)$ is the *labelling function* for nodes $(l_N : N \to \mathcal{L}_N)$ and edges $(l_E : E \to \mathcal{L}_E)$. If G is a graph, we denote by N_G its node set, by E_G its edge set, and by l_G its labelling function. An edge of the form (n, n) is called a *loop*. For an edge (n_1, n_2), n_1 and n_2 are called *end nodes* (or *endpoints*) with n_1 the *source* and n_2 the *target*; moreover we say that n_1 and n_2 are *adjacent* or *neighbouring* nodes, with n_2 neighbour of n_1. An edge is *incident* to a node if the node is one of its end nodes. An edge is *multiple* if there is another edge with the same source and target; otherwise it is *simple*. A *multigraph* is a graph allowing multiple edges and loops. An *adjacency list* for a node is given by a list of pairs consisting of a neighbour and the corresponding edge label. If a node has no neighbour then its adjacency list is empty. A *subgraph* of a graph G is a graph whose node and edge sets are subsets of those of G. A *graph morphism* assigns the nodes and edges of a given graph to the nodes and edges of another graph while preserving adjacency. In the case of labeled graphs, the node and edge labelling is also preserved.

Graph Transformation. A *graph transformation rule* $L \rightsquigarrow R$ consists of two graphs L and R called the left- and right-hand side respectively, and a partial correspondence between elements of the left-hand side and elements of the right-hand side. This correspondence is provided by some unique identifiers associated to nodes.

As presented in [17], the application of a graph transformation rule $L \rightsquigarrow R$ to a graph G, called *host graph*, produces a new graph G' according to the following steps:

1. Find a matching morphism m for L in G (hence $m(L)$ is a subgraph of G).
2. Remove the subgraph $m(L)$ from G resulting in the context graph G^-.
3. Compute the isomorphic copy of R via m.
4. Replace $m(R)$ in the context graph G^-.
5. Reconnect the two parts $m(R)$ and G^-.

The differences between various approaches for graph replacement arise mainly in the last step, depending on the mechanism chosen for establishing connections between new and old nodes. Two particular problems are handled at this stage ([11, 17]). The first one refers to whether or not noninjective matching is allowed. For example, if two different nodes L are matched to one node in the host graph, and one of the two nodes is deleted and the

other preserved, will the node in the host graph be deleted or preserved? The second problem concerns the dangling edges in the host graph which are unmatched edges with one endpoint deleted by the transformation rule. These two problematic situations are referred to as the identification and the dangling problem respectively. We will see later how we handle these points in our framework.

3 Labeled Multigraphs with Ports

Definition 1. (Labeled multigraph with ports) *A* labeled multigraph with ports *is obtained from a labeled multigraph by associating to each node a set of* ports *such that a simple edge between two nodes (not necessarily distinct since loops are allowed) is determined by two ports, one from each node. The labelling function associates to each port a name, to each node a label consisting of the unique node identifier, a name, and a set of port labels, while to each simple edge it associates the label consisting of the ordered pair of source and target port.*

We present in Fig. 1 the abstract syntax of labeled multigraphs with ports. Names can be constants or variables. We denote by $x, y, z, \ldots$ variable port names, by $a, b, c, \ldots$ constant port names, by $X, Y, Z, \ldots$ variable node names, and by $A, B, C, \ldots$ constant node names. A constant name can be mapped by a multigraph morphism only to itself. The port names for a node are pairwise distinct. A labeled multigraph with ports can be a set of nodes or a pair of a node set and an edge set. We denote by $\mathcal{V}(G)$ the set of variables occurring in G.

UIds	i	$::= Int^+$	**Nodes**	N	$::= \langle i : n \parallel P \rangle \mid N, N$
Names	n, p	$::= String$	**Edges**	E	$::= (i^\frown p, i^\frown p) \mid E, E$
Ports	P	$::= n \mid P, P$	**Multigraphs**	G	$::= N \mid N(\!\lvert E \rvert\!)$

Fig. 1. The syntax of labeled multigraphs with ports

Hereinafter we say (labeled) multigraph instead of labeled multigraph with ports if there is no risk of confusion. We illustrate in Fig. 2 two views of such a multigraph: on the left, we use the classical drawing of a labeled multigraph, while on the right, we choose to emphasize the ports. In this paper we will use the latter more suggestive representation for multigraphs.

Graph transformation rules are instantiated in this context.

Definition 2. (Multigraph rewrite rule) *A* multigraph rewrite rule *is an ordered pair of multigraphs denoted by $G_1 \rightsquigarrow G_2$ where names may be constants or variables, all node identifiers in G_1 are variables, such that $\mathcal{V}ar(G_1) \supseteq \mathcal{V}ar(G_2)$. The multigraphs G_1 and G_2 are called the left- and right-hand side of the rule respectively.*

Definition 3. (Node-substitution) *The correspondence between nodes of the left- and right-hand side of a multigraph rewrite rule $G_1 \rightsquigarrow G_2$ is a mapping $\xi : N_{G_1} \to \mathcal{P}(N_{G_2})$, where $\mathcal{P}(N_{G_2})$ is the power set of N_{G_2}, and it is called* node-substitution. *It associates for each node n in G_1:*

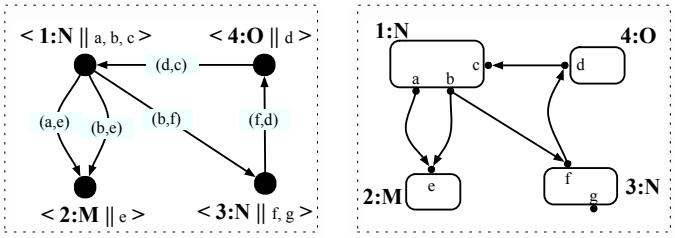


Fig. 2. Two views of a labeled multigraph with ports

- *either the set of all nodes in G_2 whose identifier is an expression (sequence of integers) containing the identifier of n,*
- *or the empty set if the identifier of n does not occur in G_2 (i.e., if the node n is deleted).*

Given the definition above, we assume that for each multigraph rewrite rule we can automatically extract from the labels the node-substitution.

Example 4. We illustrate in Fig. 3 four multigraph transformation rules: (a) splitting a node with $\xi(\langle i : X \parallel x, z\rangle) = \{\langle i.1 : X.1 \parallel x\rangle, \langle i.2 : X.2 \parallel z\rangle\}$, $\xi(\langle j : Y \parallel y\rangle) = \{\langle j : Y \parallel y\rangle\}$; (b) deleting a node with $\xi(\langle i : X \parallel x, z\rangle) = \emptyset$, $\xi(\langle j : Y \parallel y\rangle) = \{\langle j : Y \parallel y\rangle\}$; (c) deleting two edges with $\xi(N) = \{N\}$ for each node N; (d) merging two nodes $\xi(\langle i : X \parallel x, z\rangle) = \xi(\langle j : Y \parallel y\rangle) = \{\langle i.j : X.Y \parallel x, y, z\rangle\}$.

Let $L \rightsquigarrow R$ be a multigraph rewrite rule applied to the multigraph G and m a matching morphism for L in G meaning that m assigns the nodes and edges of L to partial nodes and edges in G while preserving adjacency. A *partial node* of a node N is a node with the same identifier and label, and a non-empty subset of ports of N.

The delicate point of applying $L \rightsquigarrow R$ to G is to properly define the replacement of $m(L)$ by $m(R)$ in G and the way $m(R)$ is reconnected with G. Let us first illustrate graphically in Fig. 4 the replacement procedure: the first graph is G with the dotted area representing $m(L)$; the second graph is again G where we emphasize the edges connecting unmatched nodes to matched nodes (that we call bridges); and the third graph represents the result of replacing the subgraph $m(L)$ by $m(R)$ and we see that some bridges are pending if the target formerly in $m(L)$ does no longer occur in $m(R)$. The last step of reconnecting, identifies the

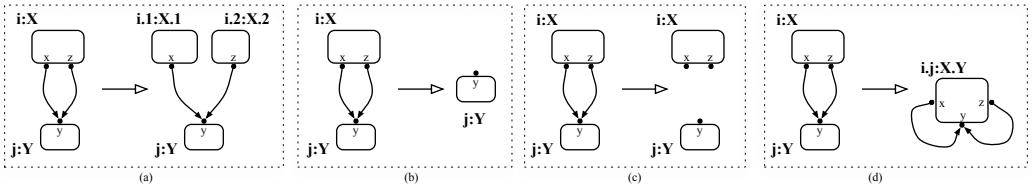


Fig. 3. Multigraph rewrite rules

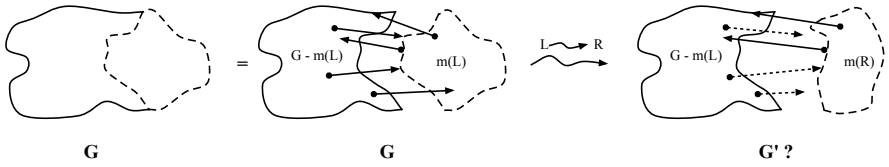


Fig. 4. A graph replacement sketch

old end node of the bridges from $m(L)$ with their correspondent nodes from $m(R)$. The same operation must be performed as well on the unmatched edges and unmatched partial nodes. Then, the result of the multigraph rewriting of G consists of putting together the context multigraph G^-, $m(R)$, and the updated unmatched edges and partial nodes, and bridges. We note that we have to update first the unmatched partial nodes, since they can be end points for unmatched edges or bridges.

In order to formalise this replacement, let us first define the context multigraph: $G^- = G \setminus m(L)$ with the set of nodes $N_{G^-} = N_G \setminus N_{m(L)} = \{\langle i : n \parallel P\rangle \in N_G \mid \nexists P' \subseteq P, P' \neq \emptyset, \text{ s.t. } \langle i : n \parallel P'\rangle \in N_{m(L)}\}$, and the set of edges $E_{G^-} = \{(i^\frown p, j^\frown r) \in E_G \mid i, j \in N_{G^-}\}$.

Let $\mathcal{U}_n$ be the set of *unmatched partial nodes*, $\mathcal{U}_n = \{\langle i : m \parallel P'\rangle \mid \langle i : m \parallel P, P'\rangle \in N_G \text{ and } \langle i : m \parallel P\rangle \in N_{m(L)}\}$, and $\mathcal{U}_e$ the set of *unmatched edges*, i.e., the not matched edges whose both endpoints are matched: $\mathcal{U}_e = \{(i^\frown p, j^\frown r) \in E_G \mid i, j \in N_{m(L)}, (i^\frown p, j^\frown r) \notin E_{m(L)}\}$.

We call *bridges* the edges from G not matched by an edge of L, with one end a matched node and the other end not matched: $\mathcal{B} = \{(i^\frown p, j^\frown r) \in E_G \mid (i \in N_{m(L)} \text{ and } j \notin N_{m(L)}) \text{ or } (i \notin N_{m(L)} \text{ and } j \in N_{m(L)})\}$.

Let ξ be the node-substitution associated to the matched multigraph rewrite rule $L \rightsquigarrow R$. If $\langle i : m \parallel P'\rangle \in \mathcal{U}_n$ and $\xi(\langle i : m \parallel P\rangle) = \{\langle i.1 : m.1 \parallel P_1\rangle, \ldots, \langle i.k : m.k \parallel P_k\rangle\}$ for $\langle i : m \parallel P\rangle \in m(L)$, then we choose a k-partition of P', say $\{P'_1, \ldots, P'_k\}$, and $\xi(\langle i : m \parallel P'\rangle) = \{\langle i.l : m \parallel P'_l\rangle \mid P'_l \neq \emptyset, l = 1..k\}$. The application of ξ on a set of edges E is defined by: $\xi(E) = \{\xi(e) \mid e \in E\}$ and $\xi((i^\frown p, j^\frown r)) = \{(i_k^\frown p, j_l^\frown r) \mid i_k \in \xi(i) \text{ s.t. } p \text{ is a port of } i_k \text{ and } j_l \in \xi(j) \text{ s.t. } r \text{ is a port of } j_l\}$, where by $\xi(i)$ we consider both the matched and the unmatched part of a node identified by i.

By analogy with term rewriting, we can write $G = G^-[m(L)]_{\mathcal{U}_n, \mathcal{U}_e, \mathcal{B}}$, as a decomposition of G into the context graph G^-, the matched subgraph $m(L)$, the unmatched partial nodes $\mathcal{U}_n$, unmatched edges $\mathcal{U}_e$, and the bridges $\mathcal{B}$. Then, once $m(R)$ is computed, it is replaced in the context multigraph G^- and the bridges are reconnected, thanks to $m(\xi)(\mathcal{U}_n)$, $m(\xi)(\mathcal{U}_e)$, and $m(\xi)(\mathcal{B})$, to get a resulting multigraph $G^-[m(R)]_{m(\xi)(\mathcal{U}_n), m(\xi)(\mathcal{U}_e), m(\xi)(\mathcal{B})}$. Note that $m(\xi)(v) = \{m(v_1), \ldots, m(v_k)\}$ if $\xi(v) = \{v_1, \ldots, v_k\}$.

We are now able to formulate the definition of multigraph rewriting.

Definition 5. (Multigraph rewriting relation) *Given a multigraph rewrite system $\mathcal{R}$, a multigraph G rewrites to a multigraph G', denoted by $G \rightsquigarrow_{\mathcal{R}} G'$, if there exists:*

- *a multigraph rewrite rule $L \rightsquigarrow R$ in $\mathcal{R}$,*
- *a multigraph morphism m such that $m(L)$ is a subgraph of G,*
- *a set of bridges $\mathcal{B}$, a set of unmatched partial nodes $\mathcal{U}_n$, and a set of unmatched edges $\mathcal{U}_e$*

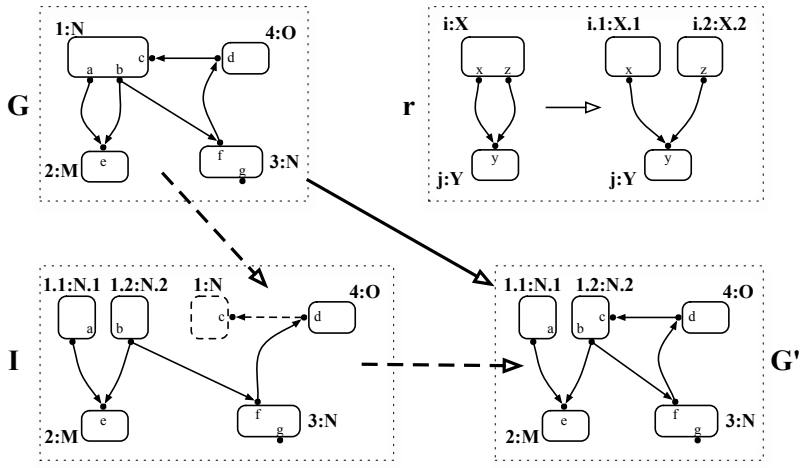


Fig. 5. An application of the multigraph rewrite rule r on G resulting in the multigraph G' with an intermediate multigraph I

such that $G = G^-[m(L)]_{\mathcal{U}_n,\mathcal{U}_e,\mathcal{B}}$ *and* $G' = G^-[m(R)]_{m(\xi)(\mathcal{U}_n),m(\xi)(\mathcal{U}_e),m(\xi)(\mathcal{B})}$.

With respect to the discussion at the end of Section 2 concerning the identification and the dangling problems, the particularities of labeled multigraphs with ports transformations are first, that we consider only injective matching morphism, second, that when deleting a node, all its incident edges are deleted as well.

Example 6. We illustrate in Fig. 5 a multigraph resulting from rewriting G (also given in Fig.2) using the rule r (also given in Fig. 3 (a)). The resulting multigraph G' is obtained by splitting the node $1 : N$, choosing to place the unmatched port c in $1.2 : N.2$, and then redirecting the two bridges $(4^\frown d, 1^\frown c)$ and $(1^\frown b, 3^\frown f)$ to $1.2 : N.2$. In the intermediate step we emphasize the incidence of the node $4 : O$ to the unmatched partial node $\langle 1 : N \parallel c\rangle$. The node-substitution may identify this partial node to either of the two resulting nodes $1.1 : N.1$ and $1.2 : N.2$, by partitioning the set of ports $\{c\}$ in two. Therefore two solutions are possible: the one illustrated in Fig. 5, and the other one where c is placed in the node $1.1 : N.1$.

In order to define multigraph rewriting in an more operational way, we choose to go in the world of algebraic terms and term rewrite rules where we take benefit from a classical ACU-matching algorithm for the application of rewrite rules.

4 Term Rewriting Semantics for Multigraph Rewriting

In this section we define an encoding function $\mathcal{E}$ for multigraphs, multigraph rewrite rules, and multigraph rewrite relation as particular terms, term rewrite rules, and term rewriting relation respectively.

$\bullet : \longrightarrow Id$	$\bullet : \longrightarrow Port$	$\bullet : \longrightarrow Node$	$\epsilon_X : \longrightarrow XSet$
$\_,\_ : XSet\ XSet \longrightarrow XSet\ [ACU(\epsilon_X)]$			
$\langle \_ : \_ \parallel \_ \rangle : Id\ Name\ PortSet \longrightarrow Node$			
$(\_,\_) : Port\ Port \longrightarrow Edge$			
$\_ \frown \_ : Id\ EdgeSet \longrightarrow Neighbour$			
$\_ \doteq \_ : Id\ NeighbourSet \longrightarrow AdjacencyEq$			
$\_(\!\|\_\|\!) : NodeSet\ AdjacencyEqSet \longrightarrow MGraph$			

Fig. 6. The set of operations $\mathcal{F}$

Concerning the problem of dangling edges, instead of mapping a node from the left-hand side to an empty set of nodes from the right-hand side, we introduce a special node $\bullet$ called the *black hole*. Consequently, we replace a deleted node in an intermediate step by a black hole whose behaviour consists in deleting itself along with the incident edges. In a similar way we use the black hole to replace in an intermediary step a deleted port as well. Hence, the dangling edges are deleted using some particular intermediate and transparent operation as we will see later in this section.

We refine the abstract syntax of multigraphs presented in Fig. 1 by an order-sorted signature $\Sigma = (\mathcal{S}, <, \mathcal{F})$ given below, where, in order to eliminate redundancies, we represent multigraphs as sets of nodes and set of adjacency lists (for each node we list its neighbours with the corresponding edges as pairs of ports):

the sort set $\mathcal{S}$ consists of sorts for each component or set of components: *Id*, *Name*, *Port*, *Node*, *Edge*, *Neighbour*, *AdjacencyEq*, *PortSet*, *NodeSet*, *EdgeSet*, *NeighbourSet*, *AdjacencyEqSet*, *MGraph*.

the subsort relation is defined by $X < XSet$ for $X \in \{Port, Node, Edge, Neighbour, AdjacencyEq, MGraph\}$, i.e. each term of sort X can be seen as a set with a single element.

the operation set $\mathcal{F}$ allowing to describe the graph structure, is given in Fig. 6 where X takes sort values from the set $\{Node, Edge, Neighbour, AdjacencyEq\}$. The associative-commutative operator $\_,\_$ (union) is overloaded on each of the set sorts, and ϵ_X denotes the identity element (the empty set) for the operation $\_,\_$. We use ϵ instead of ϵ_X whenever the sort X can be easily deduced from the context. The constant operator $\bullet$ is overloaded as well, it can be an *Id*-, a *Port*- or a *Node*-sorted term.

Let $\mathcal{X}$ be an $(\mathcal{S}, <)$-sorted family of variables.

Definition 7. (Encoding multigraphs as terms) *We encode a labeled multigraph* $G = (N, E)$ *as an algebraic term* $\mathcal{E}(G) = T_1(\!|T_2|\!)$ *of sort MGraph where:*

- $T_1 \in \mathcal{T}_{\Sigma, NodeSet}(\mathcal{X})$ *is the set of all node labels in* G*, and*
- $T_2 \in \mathcal{T}_{\Sigma, AdjacencyEqSet}(\mathcal{X})$ *is the set of adjacency equations providing the neighbours for each node in* N *(if any) and the labels corresponding to the incident edges.*

Additionally, algebraic terms encoding multigraphs must satisfy some structural properties in order to be considered well-formed.

Definition 8. (Well-formed terms) *A term* $t \in \mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$ *is* well-formed *if:*

- *each unique node identifier occurs at most once: in the node set, in the adjacency equation set as left-hand side of an adjacency equation, in the neighbour set of a node identifier;*
- *each node identifier or port occurring in the adjacency equation set must also occur in the node set.*

We also impose a canonical form (a representative of each equivalence class modulo ACU) for the terms encoding multigraphs, in order to eliminate useless information as follows:

Definition 9. (Canonical form) *A term* $t \in \mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$ *is in* canonical form *if:*

- *left-hand sides of adjacency equations are non-empty sets of neighbours;*
- *in a neighbour term, each target is connected through a non-empty set of edges to the source.*

Example 10. The multigraph G illustrated in Fig. 2 is encoded as the following term: $\mathcal{E}(G) = (\langle 1 : N \parallel a, b, c\rangle, \langle 2 : M \parallel e\rangle, \langle 3 : N \parallel f, g\rangle, \langle 4 : O \parallel d\rangle)(\!| \, 1 \doteq (2^\frown(a, e), (b, e)), (3^\frown(b, f))), \ (3 \doteq 4^\frown(f, d)), (4 \doteq 1^\frown(d, c)) \, |\!)$.

For all rewrite rules over $\mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$ we impose node identifiers occurring in the left-hand side to be variables. We say that a rewrite rule over $\mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$ is well-formed (in canonical form) if both t_1 and t_2 are well-formed (in canonical form respectively). We call *mg-rewrite rule* a well-formed rewrite rule in canonical form.

Definition 11. (Encoding multigraph rewrite rules as term rewrite rules) *Given a labeled multigraph rewrite rule* $G_1 \rightsquigarrow G_2$*, we encode it as a term rewrite rule* $\mathcal{E}(G_1 \rightsquigarrow G_2) = \mathcal{E}(G_1) \twoheadrightarrow \mathcal{E}(G_2)$.

The encoding of a multigraph rewrite rule is an mg-rewrite rule since by definition the term encoding a multigraph is well-formed and in canonical form.

The *node-substitution* from nodes of the left-hand side to nodes of the right-hand side of an mg-rewrite rule can be extracted automatically by means of an analysis on the identifier occurrences. We call this procedure GetMap and it produces a set of elementary mappings (or elementary node-substitutions) from $\mathcal{T}_{\Sigma,Node}(\mathcal{X})$ to $\mathcal{T}_{\Sigma,NodeSet}(\mathcal{X})$ for each node occurring in the left-hand side of the rule. Identity mappings are usually omitted. Note that the node-substitution for a multigraph rewrite rule is encoded as above.

Example 12. The encoding of the multigraph rewrite rule (a) given in Fig. 3 is:
$(\langle i : X \parallel x, z\rangle, \langle j : Y \parallel y\rangle)(\!|i \doteq j^\frown(x, y), (z, y)|\!) \twoheadrightarrow$
$(\langle i.1 : X.1 \parallel x\rangle, \langle i.2 : X.2 \parallel z\rangle, \langle j : Y \parallel y\rangle)(\!|(i.1 \doteq j^\frown(x, y)), (i.2 \doteq j^\frown(z, y))|\!)$
with the node-substitution $\xi = \{\langle i : X \parallel x, z\rangle \mapsto (\langle i.1 : X.1 \parallel x\rangle, \langle i.2 : X.2 \parallel z\rangle)), \ \langle j : Y \parallel y\rangle \mapsto \langle j : Y \parallel y\rangle\}$.

After encoding multigraphs and their transformation rules, we now translate multigraph rewriting into term rewriting.

In a first step we customize the rewrite rules on $\mathcal{T}_{MGraph}(\mathcal{X})$ before applying them. In order to model multigraph rewriting using algebraic terms, we need to handle the context

(Propagate)	$\{t \mapsto T\}N(\!\|A\|\!) \Longmapsto N(\!\|\{t \rightsquigarrow T\}A\|\!)$
(Distribute)	$\{t \mapsto T\}(S_1, \ldots, S_k) \Longmapsto \{t \mapsto T\}S_1, \ldots, \{t_1 \mapsto t_2\}S_k$
(ApplySrc)	$\{t \mapsto t_1, \ldots, t_n\}(i \doteq V) \Longmapsto$
	$\texttt{if } id(t) \neq i \texttt{ then } i \doteq (\{t \mapsto t_1, \ldots, t_n\}V)$
	$\texttt{ else } id(t_1) \doteq (\{t \mapsto t_1, \ldots, t_n\}V), \ldots, id(t_n) \doteq (\{t \mapsto t_1, \ldots, t_n\}V)$
(ApplyTar)	$\{t \mapsto t_1, \ldots, t_n\}(i^\frown E) \Longmapsto$
	$\texttt{if } id(t) \neq i \texttt{ then } i^\frown E \texttt{ else } id(t_1)^\frown E, \ldots, id(t_n)^\frown E$

Fig. 7. Node-substitution application

of the multigraph in which the replacement is performed. This is done thanks to the systematic enrichment of rewrite rules with extension variables that help storing the context and applying rewrite steps in subterms. This is a usual method employed when performing rewriting modulo associativity and commutativity ([15]). We usually denote by W an extension variable and by $\overline{t}$ the extension of term t. For each rewrite rule $t_1 \twoheadrightarrow t_2$, extension variables are appended to set-sorted terms to produce the extended rule $(\overline{t_1 \twoheadrightarrow t_2})$. An extension variable is added to each set-sorted subterm in the left-hand side (and accordingly in the right-hand side). This technical construction is formalised in the appendix. We just give here an example of rule extension.

Example 13. The extension of the rule given in Example 12 is:
$(\langle i : X \parallel x, z, W_1^p, W_2^p\rangle, \langle j : Y \parallel y, W_3^p\rangle, W_4^n)(\!|i \doteq (j^\frown((x, y), (z, y), W_5^e)), W_6^h), W_7^a|\!) \twoheadrightarrow$
$(\langle i.1 : X.1 \parallel x, W_1^p\rangle, \langle i.2 : X.2 \parallel z, W_2^p\rangle, \langle j : Y \parallel y, W_3^p\rangle, W_4^n)(\!|(i.1 \doteq j^\frown(x, y)),$
$(i.2 \doteq j^\frown(z, y)), (i \doteq (j^\frown W_5^e), W_6^h), W_7^a|\!)$
with the node-substitution $\xi = (\langle i : X \parallel x, z, W_1^p, W_2^p\rangle) \mapsto \langle i.1 : X.1 \parallel x, W_1^p\rangle, \langle i.2 : X.2 \parallel z, W_2^p\rangle)$, $(\langle j : Y \parallel y, W_3^p\rangle) \mapsto \{\langle j : Y \parallel y, W_3^p\rangle)$ where the extension variables W_i have appropriate set sorts. The exponents used for the extension variables indicate their set sort: p for *PortSet*, n for *NodeSet*, e for *EdgeSet*, h for *NeighbourSet*, a for *AdjacencyEqSet*.

Instantiation of a node-substitution. Let σ be a substitution and ξ a node-substitution. We denote by ξ^σ the instantiation by σ of variables occurring in ξ computed component-wise: $\{t \mapsto t_1, \ldots, t_k\}^\sigma = \{\sigma(t) \mapsto \sigma(t_1, \ldots, t_k)\}$.

Node-substitution application. The application of a node-substitution ξ on a term consists in applying sequentially each elementary node-substitution of ξ on the term and this is illustrated in Fig. 7. An elementary node-substitution is propagated inside an *MGraph*-sorted term using (Propagate), inside the set of adjacency equations of neighbours using (Distribute), and then applied on each of them. The application of a node-substitution on an adjacency equation using (ApplySrc) (or a neighbour using (ApplyTar)) transforms it in n adjacency equations (resp. neighbours), one for each corresponding node in the right-hand side of the mapping, and propagates the mapping application on the set of neighbours.

We illustrate this operation in Example 16.

After applying a node-substitution, the *MGraph*-terms may be neither well-formed, nor in canonical form, hence they require some cleaning and restructuring operations.

Cleaning. Let $\mathcal{C}$ be the rewrite system defined by the cleaning rules presented in Fig. 8 which transform terms with respect the condition of well-formedness of *MGraph*-sorted

$u, v : Id,\ t_1, t_2 : NeighbourSet,\ t_3, t_4 : EdgeSet,\ p, r : Port$
$(c_1)\ \bullet \doteq t_1 \rightarrow \epsilon_{AdjacencyEq}$
$(c_2)\ \bullet \frown t_3 \rightarrow \epsilon_{Neighbour}$
$(c_3)\ (v \doteq t_1, (u \frown t_3, (p, r), t_4), t_2) \rightarrow (v \doteq t_1, (u \frown t_3, t_4), t_2)$
$\quad$ `if` $p \notin ports(v)$ `or` $r \notin ports(u)$

Fig. 8. Cleaning rules $\mathcal{C}$

terms specified in Definition 8 as follows: (c_1) deletes the adjacency equations for black holes; (c_2) deletes the black hole neighbours; (c_3) handles the removal of extra-edges (edges whose endpoints do not appear among the ports of the connected nodes).

The extra-edges appear as a consequence of the raw application of the node-substitution according to (ApplySrc) and (ApplyTar) on adjacency equations and neighbours respectively without checking the connectivity between the new nodes. An illustration of the cleaning operation can be found in Example 16.

Restructuring. Let $\mathcal{R}$ be the rewrite system defined by the rules presented in Fig. 9 which transforms terms in the canonical form specified by Definition 9 as follows: (r_1) merges nodes having the same identifier into one node by merging their associated information; (r_2) deleted a neighbour with empty set of edges; (r_3) merges the associated sets of edges for identical neighbours; (r_4) deletes adjacency equations with empty set of neighbours; (r_5) merges adjacency equations having the same identifier in the first component into one adjacency equation by merging the sets in the second component.

$i : Id,\ v : Name,\ t_1, t_2 : PortSet,\ t_3, t_4 : NeighbourSet,\ t_5, t_6 : EdgeSet$
$(r_1)\ \langle i : v \parallel t_1 \rangle, \langle i : v \parallel t_2 \rangle \ \rightarrow\ \langle i : v \parallel t_1, t_2 \rangle$
$(r_2)\ i \frown \epsilon_{Edge} \ \rightarrow\ \epsilon_{Neighbour}$
$(r_3)\ (i \frown t_5), (i \frown t_6) \ \rightarrow\ i \frown t_5, t_6$
$(r_4)\ i \doteq \epsilon_{Neighbour} \ \rightarrow\ \epsilon_{AdjacencyEq}$
$(r_5)\ (i \doteq t_3), (i \doteq t_4) \ \rightarrow\ i \doteq t_3, t_4$

Fig. 9. Restructuring rules $\mathcal{R}$

It is not difficult to prove that:

Proposition 14. *$\mathcal{C}$ and $\mathcal{R}$ are strongly terminating and confluent.*

We are now ready to define the mg-rewriting relation. Operationally, we apply extended rewrite rules and we deal only with rule application at the top position of terms.

Definition 15. (mg-rewriting relation) *A term t of sort MGraph rewrites to a term t' using an mg-rewrite rule $r : t_1 \rightarrowtriangle t_2$ where $t_1, t_2 \in \mathcal{T}_{\Sigma, MGraph}(\mathcal{X})$ with $\overline{r} : \overline{t_1 \rightarrowtriangle t_2}$ and $\xi = \mathsf{GetMap}(\overline{r})$, which is denoted by $t \xrightarrow{\overline{r}} t'$, if there exists a substitution σ solution of the ACU-matching problem $\overline{t_1} \ll t$ such that $t' = \xi^{\sigma}(\sigma(\overline{t_2}))\downarrow_{\mathcal{C}}\downarrow_{\mathcal{R}}$. We call this relation* mg-rewriting *and we say that t* mg-rewrites *to t' by r.*

Example 16. We present here a solution of mg-rewriting the term t from the Example 10 encoding the multigraph from Fig. 2 using the rewrite rule $t_1 \twoheadrightarrow t_2$ given in Example 12 and extended in Example 13 which encodes the multigraph rewrite rule (a) from Fig. 3. This mg-rewriting corresponds to the multigraph rewriting depicted in Fig. 5.

1. one solution of the matching problem $\overline{t_1} \ll t$ is given by the substitution $\sigma = \{i \mapsto 1,\ X \mapsto N,\ x \mapsto a,\ z \mapsto b,\ j \mapsto 2,\ Y \mapsto M,\ y \mapsto e,\ W_1^p \mapsto \epsilon,\ W_2^p \mapsto c,\ W_3^p \mapsto \epsilon,\ W_4^n \mapsto (\langle 3 : N \parallel f, g\rangle, \langle 4 : O \parallel d\rangle),\ W_5^e \mapsto \epsilon,\ W_6^h \mapsto 3^\frown(b, f),\ W_7^a \mapsto ((3 \doteq 4^\frown(f, d)), (4 \doteq 1^\frown(d, c))\}$
2. $\sigma(\overline{t_2}) = (\langle 1.1 : N.1 \parallel a, \rangle, \langle 1.2 : N.2 \parallel b, c\rangle, \langle 2 : M \parallel e\rangle, \langle 3 : N \parallel f, g\rangle, \langle 4 : O \parallel d\rangle)(\!|\ 1.1 \doteq (2^\frown(a, e)),\ 1.2 \doteq (2^\frown(b, e)), (3^\frown(b, f)),\ (3 \doteq 4^\frown(f, d)), (4 \doteq 1^\frown(d, c))\ |\!)$ and we note the occurrence of the node identifier 1 in the adjacency equation set which is no longer a valid node identifier in this term;
3. $\xi^\sigma = (\langle 1 : N \parallel a, b, c\rangle) \mapsto \langle 1.1 : N.1 \parallel a\rangle, \langle 2.2 : N.2 \parallel b, c\rangle),\ (\langle 2 : M \parallel e\rangle) \mapsto \{\langle 2 : M \parallel e\rangle\}$
4. $\xi^\sigma(\sigma(\overline{t_2})) \Longrightarrow^+ (\langle 1.1 : N.1 \parallel a, \rangle, \langle 1.2 : N.2 \parallel b, c\rangle, \langle 2 : M \parallel e\rangle, \langle 3 : N \parallel f, g\rangle, \langle 4 : O \parallel d\rangle)(\!|\ 1.1 \doteq (2^\frown(a, e)),\ 1.2 \doteq (2^\frown(b, e)), (3^\frown(b, f)),\ (3 \doteq 4^\frown(f, d)), (4 \doteq 1.1^\frown(d, c), 1.2^\frown(d, c))\ |\!)$ by applying at the end the rule (ApplyTar) on $4 \doteq 1^\frown(d, c)$;
5. $\xi^\sigma(\sigma(\overline{t_2}))\ \downarrow_{\mathcal{C}} = (\langle 1.1 : N.1 \parallel a, \rangle, \langle 1.2 : N.2 \parallel b, c\rangle, \langle 2 : M \parallel e\rangle, \langle 3 : N \parallel f, g\rangle, \langle 4 : O \parallel d\rangle)(\!|\ 1.1 \doteq (2^\frown(a, e)),\ 1.2 \doteq (2^\frown(b, e)), (3^\frown(b, f)),\ (3 \doteq 4^\frown(f, d)), (4 \doteq 1.2^\frown(d, c))\ |\!)$ using the reduction $4 \doteq 1.1^\frown(d, c), 1.2^\frown(d, c) \xrightarrow{(ExtraEdges)} 4 \doteq 1.2^\frown(d, c)$ since c does not occur in the port set of the node identified by 1.1, but in the one of the node identified by 1.2;
6. $\xi^\sigma(\sigma(\overline{t_2}))\ \downarrow_{\mathcal{C}}\downarrow_{\mathcal{R}} = \xi^\sigma(\sigma(\overline{t_2}))\ \downarrow_{\mathcal{C}}$ since no restructuring rule is applied.

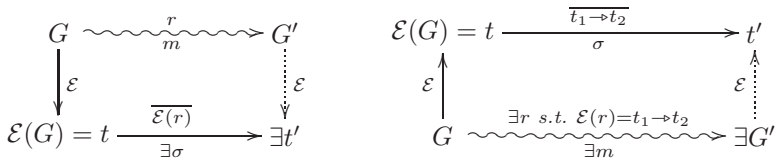
The above solution σ of the matching problem led to the result term $\xi^\sigma(\sigma(\overline{t_2}))\ \downarrow_{\mathcal{C}}\downarrow_{\mathcal{R}}$ which in fact is the encoding of the multigraph G' in Fig. 5.

Proposition 17. *If t mg-rewrites to t' and t is a well-formed term in canonical form then t' is well-formed and in canonical form.*

Theorem 18. (Operational correspondence)

1) Let G, G' be two multigraphs, r a multigraph rewrite rule and m a multigraph morphism such that $G \overset{r}{\rightsquigarrow} G'$ using m. Then there exists a substitution σ and a term t' such that $\mathcal{E}(G) \xrightarrow{\overline{\mathcal{E}(r)}} t'$ using the substitution σ and $t' = \mathcal{E}(G')$.

2) Let G be multigraph, $t = \mathcal{E}(G)$, $t_1 \twoheadrightarrow t_2$ an mg-rewrite rule, and t' such that $t \xrightarrow{\overline{t_1 \twoheadrightarrow t_2}} t'$ with σ the solution of the matching $\overline{t_1} \ll t$ used in the rewriting. Then there exist (i) a multigraph rewrite rule r satisfying $\mathcal{E}(r) = t_1 \twoheadrightarrow t_2$, (ii) a multigraph morphism that can be constructed using σ and the structures of t and G, and (iii) a multigraph G' such that $G \overset{r}{\rightsquigarrow} G'$ using the matching morphism m and $\mathcal{E}(G') = t'$.



There is a correspondence as well between all possible results of rewriting a multigraph G using a rule $G_1 \rightsquigarrow G_2$ and a morphism m, and possible results of rewriting $t = \mathcal{E}(G)$ using $t_1 \twoheadrightarrow t_2 = \mathcal{E}(G_1 \rightsquigarrow G_2)$ since all solutions of the matching $\overline{t_1} \ll t$ have as common basis the encoding of m, but different mappings for the extension variables. So, while for multigraphs the application of the node-substitution on unmatched partial nodes produces k results, the application of node-substitution for a term produces a term: for terms, the k solutions arise from different solutions of the matching problem for the extension variables.

5 A Multigraph Rewriting Calculus

We start this section with a short overview of the rewriting calculus (or ρ-calculus) and continue with presenting the embedding of mg-rewriting in the rewriting calculus, resulting in a multigraph term rewriting calculus.

5.1 The Rewriting Calculus

The rewriting calculus (or ρ-calculus) [9] extends first-order term rewriting and λ-calculus. From the λ-calculus, the ρ-calculus inherits its higher-order capabilities and the explicit treatment of functions and their applications. It was introduced to make all the basic ingredients of rewriting explicit objects, in particular the notions of rewrite rule (or abstraction) "$\_ \twoheadrightarrow \_$", rule application "$\_\_$", and set of results "$\_ \wr \_$". In the ρ-calculus, the usual λ-abstraction $\lambda x.t$ is replaced by a rule abstraction $T_1 \twoheadrightarrow T_2$, where T_1 and T_2 are two arbitrary terms, and the free variables of T_1 are bound in T_2.

The syntax is defined in Fig. 10 with $\mathcal{X}$ the set of variables and $\mathcal{K}$ the set of constants. The operator "$\_ \wr \_$" groups terms together into *structures*, and, depending on the chosen theory for this operator, it provides lists, sets or multisets to represent multiple results.

The small-step reduction semantics of the ρ-calculus is defined by the two evaluation rules in Fig. 11. If the matching problem $p \ll t_3$ has a solution σ, then the application of the rewrite rule to t_3 evaluates to $\sigma(t_2)$. The set of patterns is not a priori fixed, and the matching power of the ρ-calculus can be regulated using arbitrary theories. Therefore the semantics of the calculus depends essentially on these parameters.

An important feature of the ρ-calculus is its capability of encoding *rewrite strategies* as shown in [10]. The basic strategies are the rewrite rules. An immediate application of the use of rewrite strategies in ρ-calculus is the encoding of the conditional rewriting ([9]).

The ρ-calculus has been proved confluent for linear algebraic patterns [9].

5.2 Embedding mg-Rewriting into Rewriting Calculus

Relying on the encoding of multigraph rewriting, we can now encode the mg-rewriting in the ρ-calculus as follows:

Terms	$\mathcal{T} ::= \mathcal{X} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\mathcal{T} \mid \mathcal{T} \wr \mathcal{T}$
Patterns	$\mathcal{P} \subseteq \mathcal{T}$

Fig. 10. The syntax of ρ-calculus

$$(\rho)\ (p \twoheadrightarrow t_2)t_3 \rightarrow_\rho \sigma_1(t_2) \wr \ldots \wr \sigma_n(t_2) \wr \ldots \text{ with } \sigma_i \in \mathcal{S}ol(p \ll t_3)$$
$$(\delta)\quad (t_1 \wr t_2)t_3 \rightarrow_\delta t_1 t_3 \wr t_2 t_3$$

Fig. 11. The semantics of ρ-calculus

- take for $\mathcal{K}$ the operation symbols in $\mathcal{F}$ with the partial ordered set of sorts $(\mathcal{S}, <)$ and for $\mathcal{X}$ an $(\mathcal{S}, <)$-sorted family of variables;
- consider as patterns well-formed terms in canonical form in $\mathcal{T}_{\Sigma, MGraph}(\mathcal{X})$;
- consider only rewrite rules whose both sides are patterns.

We benefit in addition of the structure operator which allows grouping rules or results of applications.

As for the semantics, while (δ) dealing with the distributivity of the application over structures is taken as such from the ρ-calculus, we need a new rule for the application of a rewrite rule $T_1 \twoheadrightarrow T_2$ on a well-formed term T_3 in canonical form as follows:

$$(\rho_{mg})(\overline{T_1 \twoheadrightarrow T_2})\ T_3 \rightarrow_\rho S(\sigma_1(\overline{T_2})) \wr \ldots \wr S(\varsigma_n(\overline{T_2})) \wr \ldots,$$
$$\texttt{if } \sigma_i \in \mathcal{S}ol(\overline{T_1} \ll T_3), \xi = \mathsf{GetMap}(\overline{T_1 \twoheadrightarrow T_2}), \varsigma_i = \sigma_i \circ \xi^{\sigma_i}$$

where $\overline{T_1 \twoheadrightarrow T_2}$ is the extended rule associated to $T_1 \twoheadrightarrow T_2$, the matching problem is solved using an ACU-matching algorithm, and S is a strategy which reduces a term to its normal form w.r.t. the cleaning and restructuring rules respectively.

We obtain this way a *rewriting calculus for labeled multigraphs with ports*, which is an instance of ρ-calculus, and we call it the ρ_{mg}-calculus.

6 Conclusion

An implementation for the multigraphs with ports is currently developed in TOM [1] using pointers as for the termgraph implementation ([5]). We use a more efficient encoding for programming by expressing each edge by a pointer to the target port associated to the source port. Since multigraphs can be quite large, the use of pointers and TOM's maximal sharing represent an important gain in resource consumption for the implementation.

The generality of the notion of multigraph with ports allows expressing different types of multigraphs, from simple graphs to multigraphs with ports with states.

An immediate application of multigraphs with ports is for modeling protein-protein interactions concerned with the connectivity inside molecular complexes (see [12] for a process algebra approach, and [6] for an approach based in graph rewriting). Proteins are abstracted as boxes with interaction sites on the surface having particular states. Hence adding a refinement on the ports and calling them sites, the multigraph rewriting and its correspondent rewriting calculus become suitable for modelling the interactions of molecular complexes. Membranes can also form complexes, called tissues, due to the binding proteins on their surfaces. While on one side we can model interactions between biochemical entities like proteins, proteins and lipids, or membranes, on the other side we are able to model as well chemical reactions (like the ones in [2]) using the multigraph rewriting: atoms represent the

nodes, the covalence of an atom gives the number of identical ports, and chemical bonds between atoms are multiple edges.

For further applications, an interesting research direction is to enhance multigraphs with ports with a hierarchical node structure describing nested resources, and to relate them to Milner's bigraphs [16].

References

1. TOM web page.
2. O. Andrei, L. Ibanescu, and H. Kirchner. Non-intrusive Formal Methods and Strategic Rewriting for a Chemical Application. In K. Futatsugi, J.-P. Jouannaud, and J. Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*, pages 194–215. Springer, 2006.
3. O. Andrei and H. Kirchner. Strategic Rewriting for Molecular Graphs. Submitted.
4. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
5. E. Balland and P. Brauner. Term-graph rewriting in Tom using relative positions. In *Preproceedings of 4th International Workshop on Computing with Terms and Graphs - TERMGRAPH*, 2007.
6. M. L. Blinov, J. Yang, J. R. Faeder, and W. S. Hlavacek. Graph theory for rule-based modeling of biochemical networks. In *Proceedings of BioCONCUR 2005: A workshop on concurrent models in molecular biology*, 2005.
7. O. Bournez, L. Ibanescu, and H. Kirchner. From Chemical Rules to Term Rewriting. *Electronic Notes in Theoretical Computer Science*, 147(1):113–134, 2006.
8. U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall. GraphML Progress Report: Structural Layer Proposal. In *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 501–512. Springer, 2002.
9. H. Cirstea and C. Kirchner. The rewriting calculus - Part I and II. *Logic Journal of the IGPL*, 9(3), 2001.
10. H. Cirstea, C. Kirchner, L. Liquori, and B. Wack. Rewrite strategies in the rewriting calculus. *Electronic Notes in Theoretical Computer Science*, 86(4):593–624, Jun 2003.
11. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 163–246. World Scientific, 1997.
12. V. Danos and C. Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.
13. J. A. Goguen and J. Meseguer. Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theoretical Computer Science*, 105(2):217–273, 1992.
14. C. Kirchner and H. Kirchner. Rewriting, Solving, Proving. A preliminary version of a book available at `http://www.loria.fr/ ckirchne/=rsp/rsp.pdf`.
15. H. Kirchner and P.-E. Moreau. Promoting rewriting to a programming language: a compiler for non-deterministic rewrite programs in associative-commutative theories. *J. Funct. Program.*, 11(2):207–251, 2001.
16. R. Milner. Bigraphical Reactive Systems. In K. G. Larsen and M. Nielsen, editors, *CONCUR*, volume 2154 of *Lecture Notes in Computer Science*, pages 16–35. Springer, 2001.
17. A. Schürr. Programmed Graph Replacement Systems. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 479–546. World Scientific, 1997.

A Extending a mg-rewrite rule

For this operation we consider also the trivial node-substitutions, $t \mapsto t$ with $t \in \mathcal{T}_{\Sigma,Node}(\mathcal{X})$.

Definition 19. (Extending an mg-rewrite rule) *The* extension *of an mg-rewrite rule $t_1 \twoheadrightarrow t_2$, where ξ is its node-substitution, consists in adding extension variables for the set operators according to the following steps:*

1. **(Extend the node-substitution)**
   If $\xi = \{\xi_1, \ldots, \xi_m\}$ then its extension is $\overline{\xi} = \{\overline{\xi_1}, \ldots, \overline{\xi_m}\}$. For each elementary node-substitution $\xi_k = \{\langle i : t \parallel P\rangle \rightsquigarrow \langle i_1 : t_1 \parallel P_1\rangle, \ldots, \langle i_n : t_n \parallel P_n\rangle$ with $P, P_1, \ldots, P_n \in \mathcal{T}_{\Sigma,PortSet}(\mathcal{X})$, its extension is:

$$\overline{\xi_k} = \langle i : t \parallel P, W_1, \ldots, W_n\rangle \rightsquigarrow \langle i_1 : t_1 \parallel P_1, W_1\rangle, \ldots, \langle i_n : t_n \parallel P_n, W_n\rangle$$

   where $\{W_k\}_{k=1..n}$ are pairwise distinct and fresh extension variables of sort PortSet.
   If $\xi_k = \langle i : t \parallel P\rangle \rightsquigarrow \bullet$ then $\overline{\xi_k} = \langle i : t \parallel P, W\rangle \rightsquigarrow \bullet$, with W a fresh extension variable of sort PortSet.
2. **(Extend the left-hand side of the rule)**
   If $t_1 = N_1(\!|A_1|\!)$, its extension is $\overline{t_1} = \overline{N_1}(\!|\overline{A_1}|\!)$ where:
   - *$\overline{N_1}$ is obtained from N_1 by replacing each Node-term by its extension as computed for the node-substitution and appending a fresh extension variable of sort NodeSet;*
   - *$\overline{A_1}$ is obtained from A_1 by adding a new extension variable for each term of sort EdgeSet, NeighbourSet, AdjacencyEqSet.*
3. **(Extend the right-hand side of the rule)**
   If $t_2 = N_2(\!|A_2|\!)$ and $\overline{t_1} = (\overline{u_1}, \ldots, \overline{u_m}, W_1)(\!|\overline{a_1}, \ldots, \overline{a_l}, W_2|\!)$ then $\overline{t_2} = \overline{N_2}(\!|\overline{A_2}|\!)$ where:
   - *$\overline{N_2} = (\overline{\xi}(\overline{u_1}), \ldots, \overline{\xi}(\overline{u_1})) \downarrow_{\mathcal{R}}, W_1, \{\langle i : v \parallel P\rangle \in N_2 \mid i \notin dom(\xi)\}$, where $\mathcal{R}$ are the rules for transforming a term in canonical form;*
   - *$\overline{A_2} = A_2, (\overline{a_1} \setminus a_1), \ldots, (\overline{a_l} \setminus a_l)$ where $\setminus$ computes the difference between adjacency equations which consists in removing the edges appearing in the right-hand side from the left-hand side (for example $(X \doteq (Y^\frown x, L_1), L_2) \setminus (X \doteq Y^\frown x)) = (X \doteq (Y^\frown L_1), L_2))$.*

B Proofs

Proposition 20. *$\mathcal{C}$ and $\mathcal{R}$ are strongly terminating and confluent.*

Proof. For a term $t \in \mathcal{T}_\Sigma(\mathcal{X})$, we denote by $|t|$ the size of t, and, for $x \in \mathcal{X}$ by $|t|_x$ the number of occurrences of x in t. Then we define as expected a reduction order $>$ on $\mathcal{T}_\Sigma(\mathcal{X})$ by: $t > t'$ iff $|t| > |t'|$ and $\forall x \in \mathcal{X}, |t|_x > |t'|_x$, which helps proving that both rewriting systems $\mathcal{C}$ and $\mathcal{R}$ strongly terminate.

Each of the rewriting system $\mathcal{C}$ and $\mathcal{R}$ is confluent since all their critical pairs, which are quite simple to find, are joinable.

Proposition 21. *If t mg-rewrites to t' and t is a well-formed term in canonical form then t' is well-formed and in canonical form.*

Proof. The uniqueness of the occurrences of a node identifier in the node set is ensured by the normalization w.r.t. (r_1), in the adjacency equation set as left-hand side of an by the normalization w.r.t. (r_5), and in the neighbour set by the normalization w.r.t. (r_3).

Since t is well-formed, the application of a mg-rewrite rule does not introduce new node identifiers in the adjacency equation list without introducing them as well in the node set. The occurrences of ports that no longer exist (since the node they were placed in was deleted) are removed from the adjacency equation set using the rules (c_1) and (c_2). In addition, the normalization w.r.t. to the rule (c_3) ensures the exclusive presence of edges with valid ports.

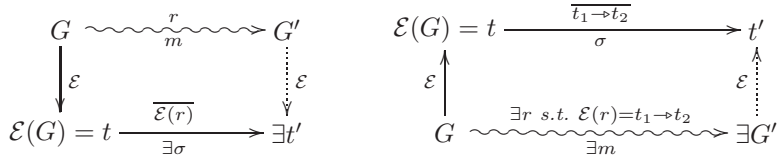
The rules (r_2) and (r_4) eliminate adjacency equations with empty sets of neighbours and neighbours with empty sets of edges.

In conclusion, since t' is irreducible w.r.t. $\mathcal{R}$ and $\mathcal{C}$ by definition, it results that t' is well-formed and in canonical form.

Theorem 22. (Operational correspondence)

1) Let G, G' be two multigraphs, r a multigraph rewrite rule and m a multigraph morphism such that $G \overset{r}{\rightsquigarrow} G'$ using m. Then there exists a substitution σ and a term t' such that $\mathcal{E}(G) \overset{\overline{\mathcal{E}(r)}}{\longrightarrow\!\!\!\!\!\rightarrow} t'$ using the substitution σ and $t' = \mathcal{E}(G')$.

2) Let G be multigraph, $t = \mathcal{E}(G)$, $t_1 \twoheadrightarrow t_2$ an mg-rewrite rule, and t' such that $t \overset{\overline{t_1 \twoheadrightarrow t_2}}{\longrightarrow\!\!\!\!\!\rightarrow} t'$ with σ the solution of the matching $\overline{t_1} \ll t$ used in the rewriting. Then there exist (i) a multigraph rewrite rule r satisfying $\mathcal{E}(r) = t_1 \twoheadrightarrow t_2$, (ii) a multigraph morphism that can be constructed using σ and the structures of t and G, and (iii) a multigraph G' such that $G \overset{r}{\rightsquigarrow} G'$ using the matching morphism m and $\mathcal{E}(G') = t'$.



Proof. 1) We start by encoding the multigraph rewrite rule $r = G_1 \rightsquigarrow G_2$ as $\mathcal{E}(r) = t_1 \twoheadrightarrow t_2$ and extending it.

Let us compute a substitution σ' based on the matching morphism m. For each mapping of the type $m(\langle i : n \parallel P \rangle) = \langle i' : n' \parallel P' \rangle$ we define $\sigma'(i) = i'$, and if n is variable then $\sigma'(n) = n'$. While for each mapping $m(p) = p'$ with $p, p' : Port$, if p is variable then $\sigma'(p) = p'$.

At this point $\sigma'(\overline{t_1})$ contains as variables only extension variables which capture sets of nodes, ports, adjacency equations, neighbours, and edges. By solving the matching problem $\sigma'(\overline{t_1}) \ll t$ we recover a substitution σ'' for the extension variables. During the solving process of the matching problem $\sigma'(\overline{t_1}) \ll t$, we replace a matching equation $u \ll v$ with $u, v \in \mathcal{T}_{\Sigma, Node}(\mathcal{X})$ by $\xi(u) \ll \xi(v)$ and solve it. This ensures us to choose the same partition for the port sets as for the unmatched partial nodes in the multigraph rewriting process.

Let us define σ as the composition of σ' and σ''. Then the mapping ξ^σ is built to mimic the connection process between nodes of G^- and $m(G_2)$ by handling the unmatched partial nodes, the unmatched edges and the bridges. The unmatched partial nodes are replaced by their correspondents, the endpoints of unmatched edges are updated, and the dangling bridges are redirected by applying ξ^σ to $\sigma(\overline{t_2})$. Consequently, $\xi^\sigma(\sigma(\overline{t_2}))$ is a representation

of $G^-[m(G_2)]_{m(\xi)(\mathcal{U}_n),m(\xi)(\mathcal{U}_e),m(\xi)(\mathcal{B})}$. Then normalization w.r.t. the cleaning rules of $\mathcal{C}$ corresponds to removing deleted nodes and ports and their incident edges. In the end, we make sure the term is well-formed by normalizing w.r.t. $\mathcal{R}$. We thus get a term t' which is indeed $\mathcal{E}(G')$.

2) Let us first define recursively the inverse operation of the encoding $\mathcal{E}$, which we denote by $\mathcal{E}^{-1}$:

$\mathcal{E}^{-1}(N(\!|A|\!)) = \mathcal{E}^{-1}(N)(\!|\mathcal{E}^{-1}(A)|\!)$, if $N(\!|A|\!)$ is well-formed and in canonical form
$\mathcal{E}^{-1}(T_1, \ldots, T_k) = \{\mathcal{E}^{-1}(T_1), \ldots, \mathcal{E}^{-1}(T_k)\}$, for $T_1, \ldots, T_k$ a set-like term
$\mathcal{E}^{-1}(\langle i : n \parallel P\rangle = \langle i : n \parallel \mathcal{E}^{-1}(P)\rangle$
$\mathcal{E}^{-1}(p) = p$
$\mathcal{E}^{-1}(i \doteq N_1, \ldots, N_k) = \mathcal{E}^{-1}(i \doteq N_1) \cup \mathcal{E}^{-1}(i \doteq N_k)$
$\mathcal{E}^{-1}(i \doteq j^\frown(p_1, r_1), \ldots, (p_k, r_k)) = \{(i^\frown p_1, j^\frown r_1), \ldots, (i^\frown p_k, j^\frown r_k)\}$
$\mathcal{E}^{-1}(t_1 \twoheadrightarrow t_2) = \mathcal{E}^{-1}(t_1) \rightsquigarrow \mathcal{E}^{-1}(t_2)$

Let $G_1 \rightsquigarrow G_2 = \mathcal{E}^{-1}(t_1 \twoheadrightarrow t_2)$, and let σ' be the restriction of σ on non-extension variables (hence it maps variable identifiers, node and port names). If we consider the set of nodes encoded by t_1, the substitution σ' and the identity mapping for the rest of the names not in the domain of σ', we are able to construct the corresponding matching morphism for G_1 and G.

Let t'_1 be a term obtained from $\overline{t_1}$ by keeping only the structure along with the identifiers, node names, and extension variables; then, once they are instantiated by σ:

- the subterms $\langle i : n \parallel W\rangle$ encode unmatched partial nodes,
- the extension variable of sort *NodeSet* encodes the context nodes,
- the subterms $i \doteq (j^\frown W_1), W_2$ encode unmatched edges and bridges if $i, j \neq \bullet$,
- the extension variable of sort *NeighbourSet* encodes bridges and context edges.

Since $\mathcal{E}^{-1}(\sigma(\overline{t_1})) = G$ and $\mathcal{E}^{-1}(\sigma(t_1)) = m(G_1)$, we saw above that G is a composition of $m(G_1)$, unmatched partial nodes and edges, bridges, and a context graph. Hence we can write $G = G^-[m(G_1)]_{\mathcal{U}_n,\mathcal{U}_e,\mathcal{B}}$.

The ACU-matching algorithm returns all solutions for the extension variables corresponding to all partitions of the port sets in the unmatched partial nodes. Since among all solutions, the substitution σ is chosen, then ξ^σ provides the right partition of the port sets needed for updating the unmatched partial nodes via the node-substitution in the multigraph rewriting. Hence we obtain a resulting graph $G' = G^-[m(R)]_{m(\xi)(\mathcal{U}_n),m(\xi)(\mathcal{U}_e),m(\xi)(\mathcal{B})}$ which is encoded by t'.