Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science 147 (2006) 113–134
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

From Chemical Rules to Term Rewriting [1]

Olivier Bournez[2] Liliana Ibănescu[3] Hélène Kirchner[4]

LORIA-INRIA & LORIA-UHP & LORIA-CNRS
Campus scientifique BP 239
F-54506 Vandoeuvre-lès-Nancy Cedex, France

Abstract

In this paper, rule-based programming is explored in the field of automated generation of chemical reaction mechanisms. We explore a class of graphs and a graph rewriting relation where vertices are preserved and only edges are changed. We show how to represent cyclic labeled graphs by decorated labeled trees or forests, then how to transform trees into terms. A graph rewriting relation is defined, then simulated by a tree rewriting relation, which can be in turn simulated by a rewriting relation on equivalence classes of terms. As a consequence, this kind of graph rewriting can be implemented using term rewriting. This study is motivated by the design of the GasEl system for the generation of kinetics reactions mechanisms. In GasEl, chemical reactions correspond to graph rewrite rules and are implemented by conditional rewriting rules in ELAN. The control of their application is done through the ELAN strategy language.

Keywords: term and graph rewriting, rule-based programming, strategy language, cyclic labeled graph, automated generation of chemical reaction mechanisms.

1 Introduction

In the context of an industrial application, we have explored the use of rule-based systems and strategies, for a complex problem of chemical kinetics: the automated generation of reaction mechanisms [22,7]. In this application, called GasEl, the representation of the chemical species uses the notion of molecular

[1] We thank all participants in the initial GasEl project: members from PROTHEO team, chemists from DCPR, Nancy and Peugeot Citroën Automobiles that financially supported this project.
[2] Email: Olivier.Bournez@loria.fr
[3] Email: Mariana-Liliana.Ibanescu@loria.fr
[4] Email: Helene.Kirchner@loria.fr

doi:10.1016/j.entcs.2005.06.040

graph [12], encoded by a term structure called GasEl term introduced in [5] and studied in [15]. Chemical solutions are modeled as multisets of terms similarly to [13]. Chemical reactions are expressed by rewriting rules on molecular graphs, encoded by a set of conditional rewriting rules on GasEl terms [15]. The control of the chemical reactions chaining is easy to describe [5,6,15] using a strategy language, such as the one offered by the ELAN system [4,16].

In [5] we briefly present the problem of generation of kinetics mechanisms in the whole context of its use by chemists and industrial partners. We give a description of the chemical problem complexity and a description of the computational problems. More details are presented in [15] that extensively describes the implemented prototype and analyses the qualitative validations performed with the chemists. However the precise description of transforming chemical rules into rewriting rules was not formally given in [15], neither the detailed application of a chemical rule on a molecular solution, nor its formal correspondence with term rewriting. From this point of view, this article is an extension of the results presented in chapter 8 of [15].

In this paper, we propose a class of labeled graphs and a graph rewriting relation well-suited to applications where the sets of vertices are preserved through transformations. This is also a special kind of hyperedge rewriting [10,11,14]. We show precisely how to perform this kind of graph rewriting by encoding the graph structure by terms and the graph rewriting relation by rewriting modulo a congruence. This is performed in two steps, first by encoding cyclic labeled graphs into decorated labeled trees or forests, then by transforming trees into terms. The first transformation relies on the operation of cutting edges in cycles, which leads to the notions of hidden and revealed edges [15]. The second transformation amounts to choosing a root in the tree and ordering the immediate subgraphs of each vertex. We prove that each labeled graph rewriting step is correctly simulated by a term rewriting step modulo an equivalence relation on terms, where all terms in the same equivalence class are a representation of the same graph (up to graph isomorphism). This result justifies the GasEl implementation of chemical rewriting [15], but is general enough to apply in different contexts.

Implementation of graph rewriting by term rewriting has already been explored by several authors [3,9,19,21]. Usually labeled graphs are axiomatized equationally in such a way that graph rewriting becomes rewriting modulo the axioms. In particular, in [18,19] it is proposed to implement graph rewriting by multiset rewriting where multisets represent graphs by their adjacency list [8]. As observed in [19], this is similar in some respects to the algebraic axiomatization of Raoult and Voisin [21]. The implementation proposed in this paper, based on what we did in the GasEl project, is in some sense a higher level implementation: graphs are not implemented as adjacency lists, but are rather encoded by terms, so that more structure is preserved.

Section 2 defines the classes of decorated labeled graphs and forests, and shows how to transform labeled graphs into decorated labeled forests. A rewriting relation on these structures is defined in Section 3. Section 4 shows how to transform trees into terms and the rewriting relation on trees into term rewriting. Section 5 briefly states how these concepts have been implemented in GasEl.

2 Decorated labeled graphs and forests

A molecular graph [12] is a vertex-labeled and edge-labeled graph, where each vertex is labeled with an atom and each edge is labeled with the bond type, as given in Figure 1.

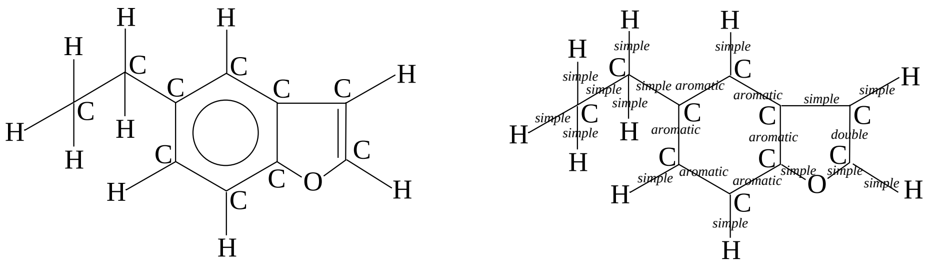


Fig. 1. A molecular graph

A chemical reaction is expressed as the application of a rewriting rule for molecular graphs. The chemical reaction from Figure 3 is an example of applying the rewriting rule from Figure 2 that breaks a simple bond between two atoms of carbon.

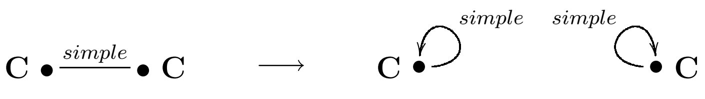


Fig. 2. Rewriting rule: breaking a simple bond

Our concern is to describe how to implement the labeled graph rewriting relation by a term rewriting relation. We proceed in two steps: first, we transform graphs into trees, or more precisely into forests, mainly by cutting cycles and introducing implicit edges also called hidden edges. This transformation is defined in this section. In the second stage, we will transform trees into terms by choosing a root and ordering direct subtrees at each vertex.

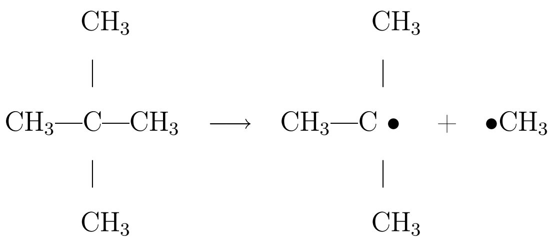


Fig. 3. A chemical reaction

2.1 Decorated labeled graphs

Let $\mathcal{L}_V$ and $\mathcal{L}_E$ be disjoint sets of labels respectively for vertices and edges. A decorated labeled graph is a labeled graph with a specific set of vertex labels: new labels are introduced to handle the fact that some cycles are cut but yet present and encoded through implicit edges, each of them being numbered by a set of labels $\mathcal{L}_C$. The label of a vertex belongs to the set $\mathcal{L}'_V = \mathcal{L}_V \times \mathcal{P}(\mathcal{L}_C \times \mathcal{L}_E)$.

Definition 2.1 [Decorated labeled graph] A *decorated labeled graph* over $\mathcal{L} = \mathcal{L}_V \cup \mathcal{L}_E \cup \mathcal{L}_C$ is a structure $G = (V, E, lab)$, such that

(i) (V, E) is a graph: V is the set of vertices and $E \subseteq V \times V$ is the set of edges;

(ii) the function $lab : V \cup E \longrightarrow \mathcal{L}'_V \cup \mathcal{L}_E$ gives
- the labels for every edge of the graph: $lab(e) \in \mathcal{L}_E, \forall e \in E$, and
- the labels for every vertex of the graph: $lab(v) \in \mathcal{L}'_V, \forall v \in V$.
  The first element of $lab(v)$ is from $\mathcal{L}_V$ and is called *vertex_ name*. The second element is a set of pairs from $\mathcal{L}_C \times \mathcal{L}_E$, denoted P_v and called *implicit_ edges*.

The set of decorated labeled graphs over $\mathcal{L}$ is denoted by $\mathcal{G}(\mathcal{L})$. When V needs to be explicitly given, it is denoted by $\mathcal{G}_V(\mathcal{L})$.

Example 2.2 In the molecular graph from Figure 1 two edges are transformed into implicit edges: (i) edge {6,11} labeled with *simple* is hidden and the representation from Figure 4 is obtained; (ii) edge {5,6} labeled with *aromatic* is hidden and the result is given in Figure 5. Hydrogen atoms are not represented.
The second graph from Figure 5 is a decorated labeled graph, where

- $\mathcal{L}_V = \{\mathbf{C}, \mathbf{H}, \mathbf{N}, \mathbf{O}, \mathbf{Cl}, \ldots\}$,
- $\mathcal{L}_E = \{simple, double, triple, aromatic\}$,
- $\mathcal{L}_C = \{1, 2, \ldots\}$,

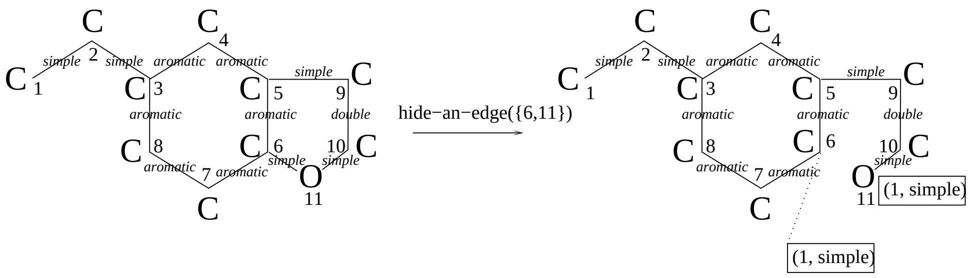


Fig. 4. Hiding edge {6,11} and encoding it by labels (1, simple) on vertices 6 and 11

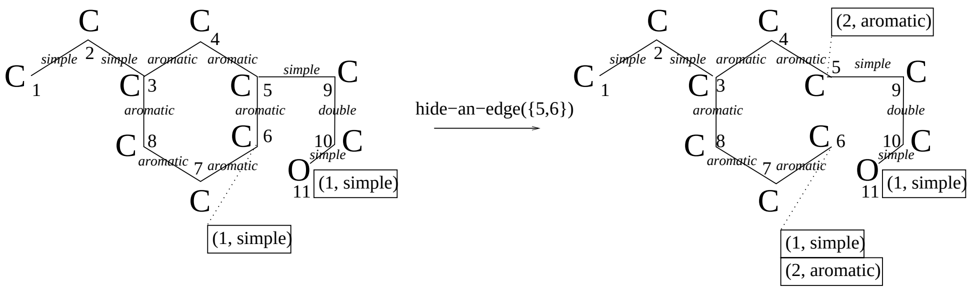


Fig. 5. Hiding edge {5,6} and encoding it by labels (2, aromatic) on vertices 5 and 6

Let $(\mathbf{C},\ P_v) = (\mathbf{C},\ \{(1,\ \text{simple}),\ (2,\ \text{aromatic})\})$ be the vertex label for node $v = 6$ in the second decorated labeled graph given in Figure 5:

- the $vertex\_name$ of $v = 6$ is $\mathbf{C}$;
- P_v is the set of $implicit\_edges$ of $v = 6$, namely {(1, simple),(2, aromatic)}:
  - · (1, simple) is a pair occurring in P_6 and P_{11} and thus encoding a hidden edge between vertices 6 and 11;
  - · (2, aromatic) is a pair occurring in P_5 and P_6 and thus encoding a hidden edge between vertices 5 and 6.

The labeled graph given in Figure 1 may be considered as an empty decorated labeled graph, i.e. a decorated labeled graph without labels for implicit edges. In the following we do not make distinction between a labeled graph and its corresponding empty decorated labeled graph.

Operations of hiding and revealing an edge are now formally described. They are internal transformations in the set of decorated labeled graphs.

Definition 2.3 [Hide-an-edge] Let $G = (V, E, lab)$ be a decorated labeled graph over $\mathcal{L}$, c be a fresh label from $\mathcal{L}_C$, $\{u, v\} \in E$ an edge of G, and $lab(\{u, v\}) = l_{uv}$. The decorated labeled graph $G_1 = (V, E_1, lab_1)$ given by

(i) $E_1 = E - \{\{u, v\}\}$,

(ii) $lab_1(e) = lab(e), \forall e \in E_1$,

(iii) $lab_1(w) = lab(w)$, for $w \in V, w \neq u, w \neq v$,

(iv) $lab_1(w) = (l_w, P_w \cup \{(c, l_{uv})\})$ if ($w = u$ or $w = v$) and $lab(w) = (l_w, P_w)$.

is obtained from G by deleting the edge $\{u, v\}$ of the graph G and encoding this edge in the labeling of the graph. The edge $\{u, v\}$ is *hidden* in the graph G_1.

Definition 2.4 [Hidden edges] Let $G = (V, E, lab)$ be a decorated labeled graph over $\mathcal{L}$. The set H of *hidden edges* of the decorated labeled graph G is the following:

$$H = \{\{u, v\} \mid \exists (c, l_{uv}) \in P_u \cap P_v\}.$$

Example 2.5 In Figure 5 there is one hidden edge in the first decorated labeled graph, and there are two hidden edges in the second.

Conversely, hidden edges may be revealed when necessary.

Definition 2.6 [Reveal-an-edge] Let $G_1 = (V, E_1, lab_1)$ be a decorated labeled graph over $\mathcal{L}$, $u, v \in V$ such that $\{u, v\} \in H$. The decorated labeled graph $G = (V, E, lab)$ given by:

(i) $E = E_1 \cup \{\{u, v\}\}$,

(ii) $lab(\{\{u, v\}\}) = l_{uv}$,

(iii) $lab(e) = lab_1(e), \forall e \in E_1$,

(iv) $lab(w) = lab_1(w)$, for $w \in V, w \neq u, w \neq v$,

(v) $lab(u) = (l_u, P_u)$, $lab(v) = (l_v, P_v)$ where $lab_1(u) = (l_u, P_u \cup \{(c, l_{uv})\}$ and $lab_1(v) = (l_v, P_v \cup \{(c, l_{uv})\}$

is obtained from G_1 by adding an edge $\{\{u, v\}\}$ to the graph G and making explicit this edge encoded in the labeling of G_1.

In the following, we only consider decorated labeled graphs that have been obtained from (possibly cyclic) labeled graphs by cutting edges. This class is characterized using the following well-formedness notion.

Definition 2.7 [Well-formed decorated labeled graph] A decorated labeled graph $G = (V, E, lab)$ over $\mathcal{L}$ is a *well-formed* decorated labeled graph if the repeated application of the reveal-an-edge operation leads to an empty decorated labeled graph.

Example 2.8 The decorated labeled graphs given in Figure 5 are well-formed decorated labeled graphs.

From now on, we suppose that all decorated labeled graphs are well-formed, unless otherwise specified.

Let $G_1 = (V, E_1, lab_1)$ be a well-formed decorated labeled graph over $\mathcal{L}$, and $G = (V, E, lab)$ be the (empty decorated) labeled graph resulting from the repeated application of the reveal-an-edge operation. The function transforming a well-formed decorated labeled graph G_1 into a labeled graph G is denoted by `decoGraph2graph`:

$$\texttt{decoGraph2graph}(G_1) = G.$$

The function `decoGraph2graph` is surjective but not injective. Two decorated labeled graphs over a fixed set of vertices that produce the same labeled graph by repeated application of revealing hidden edges and removing empty decorations, can be considered as equivalent for the following relation:

Definition 2.9 [Equivalent decorated labeled graphs] Let G_1 and G_2 be two decorated labeled graphs of $\mathcal{G}_V(\mathcal{L})$. G_1 and G_2 are equivalent, denoted $G_1 \equiv G_2$, if $\texttt{decoGraph2graph}(G_1) = \texttt{decoGraph2graph}(G_2)$. We denote the equivalence class of G_1 by $[G_1]_\equiv$.

2.2 Decorated labeled forests

The purpose of the previous formal definitions is to describe how to transform cyclic graphs into trees or forests by hiding edges and maintaining the same number of connected components.

Definition 2.10 [Decorated labeled forest] A decorated labeled graph $G = (V, E, lab)$ over $\mathcal{L}$ is called a *decorated labeled forest*, if the underlying graph (V, E) is a forest, i.e. if it has no cycle. It is a *decorated labeled tree*, if the underlying graph (V, E) is a tree, i.e. it is connected and has no cycle.

Example 2.11 The decorated labeled graph given in Figure 5 is a decorated labeled tree.

Definition 2.12 [`forest2graph`] Let F be a decorated labeled forest over $\mathcal{L}$. In this specific case, the function `decoGraph2graph` (revealing all hidden edges) is denoted `forest2graph`.

If the operation hide-an-edge given in Definition 2.3 is applied repeatedly until all edges are hidden, we get something which is similar to an adjacency list [8]. Seen as a multiset of vertices, this also corresponds to the representation of graphs proposed in [19]. Here, we do not perform this operation until all edges are hidden, but only until no cycle remains.

Definition 2.13 [`graph2forest`] Let $G = (V, E, lab)$ be a labeled graph over $\mathcal{L}$ and H be a subset of edges. The operation transforming G into a decorated labeled forest F, hiding the edges from the set $H \subseteq E$, is denoted by `graph2forest`:

$$F \in \texttt{graph2forest}(G, H).$$

Let us remark that `graph2forest` is a relation, but is not a function. However if the set H is ordered, if the set $\mathcal{L}_C$ of labels for implicit edges is fixed and if the operations hide-an-edge are applied in the order given on H, then `graph2forest` becomes a function. Let us notice also that the support graph of F, the decorated labeled forest, is a spanning forest of the graph G.

The following diagram summarizes the different transformations:

$$\begin{tikzcd} G \arrow[d, shift left, "graph2forest"] & \text{(empty decorated) labeled graph} \\ F \arrow[u, shift left, "forest2graph"] & \text{decorated labeled forest} \end{tikzcd}$$

Notice that a (connected) graph may be represented by an exponential number of decorated labeled trees or forests: let us assume that the (connected) graph has m edges and that no more than κ edges are hidden, where κ is the cyclomatic number (the minimal number of edges to be hidden to get a tree or a forest). Then the graph corresponds to at most $m!/(\kappa!(m-\kappa)!) = O(2^m)$ decorated labeled trees or forests: we need to choose κ hidden edges among the m edges. Notice that, it we consider connected graphs whose cyclomatic number κ stay bounded by some constant κ_0, this number of decorated labeled trees is polynomial in m (in $O(m^{\kappa_0})$).

The hide-an-edge operation gives a partial order on decorated labeled forests: if F_2 is obtained from F_1 by hiding an edge, then $F_1 < F_2$ (and $H_1 \subset H_2$). Dually, F_1 is obtained from F_2 by revealing the edge.

Using this relation we define the notion of *minimally hidden forest*, corresponding to a forest F obtained by hiding in a graph G the smallest number of edges, in order to get in F the same number of connected components as in G (which is equivalent to have a maximal number of revealed edges).

Definition 2.14 [Minimally hidden forest] Let $G = (V, E, lab)$ be a labeled graph, and $F \in \texttt{graph2forest}(G, H)$. F is a *minimally hidden forest* for G if F is minimal with respect to the partial order on decorated labeled forests given by the hide-an-edge operation.

F is not a *minimally hidden forest* for G if there exists a hidden edge that can be revealed in order to get a decorated labeled graph which is still a forest (no cycle is formed): in other words, $\exists F_1 : F_1 < F$.

Informally, extending the order given by hide-an-edge operation to decorated labeled graphs and starting from a graph G one can hide edges in order to cut cycles, until a forest is obtained. Indeed this is a minimal one, since hiding more edges gives again a forest. This process is depicted as follows to get F_{min}, the minimally hidden forest:

$$\begin{array}{ll} G < G_1 < \cdots < & G_\kappa \\ & \;= \\ & \underbrace{F_\kappa}_{F_{min}} < F_{\kappa+1} < \cdots < F_m \end{array}$$

We define a transformation on forests in order to obtain minimally hidden forests.

Definition 2.15 [`merge`] Let F be a decorated labeled forest. If F is not a minimally hidden forest (there exists a hidden edge that can be revealed in order to get a smaller forest) then one such hidden edge is revealed. This operation of revealing hidden edges is applied repeatedly until a minimally hidden forest F_{min} is obtained. We denote this transformation by `merge`:

$$F \mapsto_{\texttt{merge}} F_{min}$$

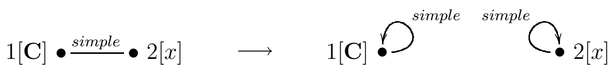
Proposition 2.16 *If $F \mapsto_{\texttt{merge}} F_{min}$ then $F_{min} < F$.*

3 Rewriting decorated labeled graphs

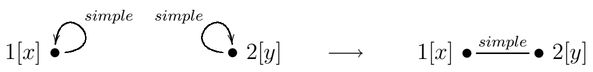
In this section, we define a rewrite relation where the set of vertices is unchanged and only edges are removed or added. In the context of chemistry, this is natural and motivated by the chemical law of the conservation of the mass. However the point of view adopted here is abstract and not restricted to chemical rewriting. This is also known and studied as hyperedge rewriting [10,14,11].

Example 3.1 The following rules encode generic chemical reactions of hydrocarbon molecules, in a specific range of temperature (oxidizing pyrolysis) [15]:

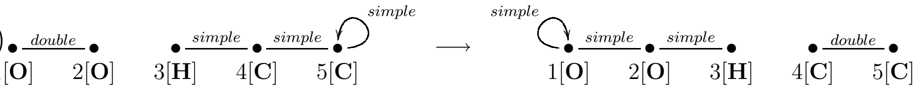
(i) unimolecular initiation (ui)

(1)


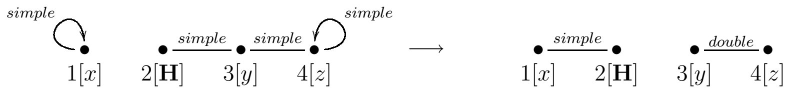
(ii) combination (co)

(2)


(iii) oxidation (ox)

(3) 

(iv) disproportionation (di)

(4) 

3.1 Rules for decorated labeled graphs

Let $\mathcal{X}_V$ and $\mathcal{X}_E$ be sets of variables for respectively vertices and edges labels, and $\mathcal{X}_C$ be a set of variables for implicit edge labels; let $\mathcal{X} = \mathcal{X}_V \cup \mathcal{X}_E \cup \mathcal{X}_C$. A decorated labeled graph with variables is a decorated labeled graph (as in Definition 2.1) such that each label of a vertex belongs to the set $(\mathcal{L}_V \cup \mathcal{X}_V) \times \mathcal{P}((\mathcal{L}_C \cup \mathcal{X}_C) \times (\mathcal{L}_E \cup \mathcal{X}_E))$ and each label of an edge belongs to $\mathcal{L}_E \times \mathcal{X}_E$. The set of decorated labeled graphs with variables is denoted by $\mathcal{G}(\mathcal{L}, \mathcal{X})$.

A decorated labeled graph with no variable, i.e. a decorated labeled graph belonging to $\mathcal{G}(\mathcal{L})$, is said to be *closed*.

Definition 3.2 [Label substitution] Let $G \in \mathcal{G}_V(\mathcal{L}, \mathcal{X})$ be a decorated labeled graph with variables. A *label substitution* σ is an application from $\mathcal{X}_V$ to $\mathcal{L}_V$, from $\mathcal{X}_E$ to $\mathcal{L}_E$ and from $\mathcal{X}_C$ to $\mathcal{L}_C$. The result of applying the label substitution σ to the labeled graph G, denoted σG, is the closed labeled graph over $\mathcal{L}$ where each vertex, edge or implicit edge variable x is replaced by $\sigma(x)$.

A rewriting rule is a pair of decorated labeled graphs with variables, as for instance those given in example 3.1. We require that the set of the vertices of both graphs coincide, and that every vertex has the same *vertex_name* in both sides.

Definition 3.3 [Decorated labeled graph rewriting rule] A *decorated graph rewriting rule* is an oriented pair of decorated labeled graphs with variables, denoted $g_l \to g_r$, such that $g_l, g_r \in \mathcal{G}_V(\mathcal{L}, \mathcal{X})$ for some V and such that vertices have the same vertex_name in g_l and g_r.

As usual, g_l and g_r are respectively called left and right-hand side of the rule.

Definition 3.4 [Decorated labeled graph rewriting system] A *decorated labeled graph rewriting system* is a set $\mathcal{RG}$ of decorated labeled graph rewriting rules.

3.2 Rewriting relation for decorated labeled graphs

Let us now turn to the definition of the rewriting relation on decorated labeled graphs. We need to precisely define the following steps: isolate a subgraph, find a substitution and a morphism from the left-hand side of a rule to this subgraph, instantiate the right-hand side by the substitution and plug it into the context. The usual notions of subgraphs and morphisms must be adapted in our context to take into account implicit edges. The key is simply to define the notion of connected vertices, just by saying that two vertices are connected either directly by an edge or indirectly by a hidden edge.

Definition 3.5 [Connected vertices] Let $G = (V, E, lab)$ be a decorated labeled graph, with hidden edges H. The vertices $u, v \in V$ are connected in G if either

- $\{u, v\} \in E$, or
- $\{u, v\} \in H$, i.e. $\exists (c, l_{uv}) \in P_u \cap\ P_v$, $c \in \mathcal{L}_C, l_{uv} \in \mathcal{L}_E$.

We denote this relation by $\texttt{connected}_G(u, v)$. When $u, v \in V$ are connected in G, $label_G(u, v)$ is defined as $lab(\{u, v\})$ in the first case, and l_{uv} in the second case.

Definition 3.6 [Subgraph of a decorated labeled graph] A decorated labeled graph $G_1 = (V_1, E_1, lab_1)$ is a *subgraph* of a decorated labeled graph $G = (V, E, lab)$, denoted by $G_1 \sqsubseteq G$, if $\texttt{decoGraph2graph}(G_1)$ is a subgraph[5] of $\texttt{decoGraph2graph}(G)$.

The decorated labeled graph decomposition in two subgraphs is based on an edge partition for both the explicit and implicit edges.

Definition 3.7 [Decorated graph decomposition in two subgraphs] Let $G = (V, E, lab)$ be a decorated labeled graph, $E = E_1 \cup E_2$ an edge partition and $H = H_1 \cup H_2$ a hidden edge partition. The graph G is decomposed in two decorated labeled subgraphs, $G_1 = (V, E_1, lab_1)$ and $G_2 = (V, E_2, lab_2)$, such that

- $vertex\_name_{G_1}(v) = vertex\_name_{G_2}(v) = vertex\_name_G(v), \forall v \in V$,
- $P_u^1 = \{(c, l_{uv}) \in P_u \mid \{u, v\} \in H_1\}$, and
- $P_u^2 = \{(c, l_{uv}) \in P_u \mid \{u, v\} \in H_2\}$,

where P_u^i is denoting the set of $implicit\_edges$ of u in G_i, for $i = 1, 2$.
We denote this decomposition by $G = G_1 \odot G_2$.

[5] The notion of subgraph is the usual component-wise inclusion.

Example 3.8 In Figure 6 is given a decomposition of a simple graph G where $E_1 = \{\{1,2\}\}$, $E_2 = \{\{1,3\}\}$ and $H_1 = \emptyset$, $H_2 = H = \{\{2,3\}\}$. When edge partitions are $E_1 = \emptyset$, $E_2 = E = \{\{1,2\},\{1,3\}\}$ and $H_1 = H = \{\{2,3\}\}$, $H_2 = \emptyset$ the same graph is decomposed as given in Figure 7.

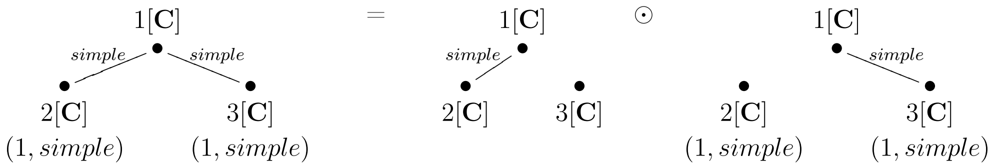


Fig. 6. Decomposition when $E = \{\{1,2\}\} \cup \{\{1,3\}\}$ and $H = \emptyset \cup \{\{2,3\}\}$

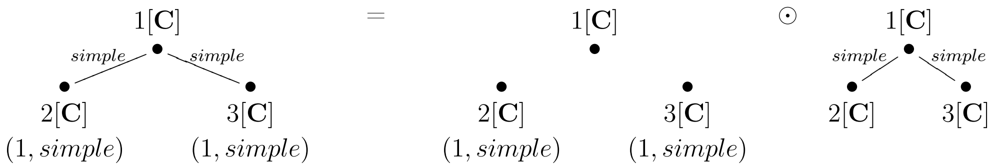


Fig. 7. Decomposition when $E = \emptyset \cup \{\{1,2\},\{1,3\}\}$ and $H = \{\{2,3\}\} \cup \emptyset$

A decorated labeled graph morphism preserves vertices and connected vertices.

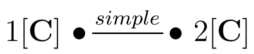
Definition 3.9 [Decorated labeled graph morphism] Let $G_1 = (V_1, E_1, lab_1)$ and $G_2 = (V_2, E_2, lab_2)$ be two decorated labeled graphs. A *morphism* from G_1 to G_2 is an injective function $f : V_1 \longrightarrow V_2$ such that:

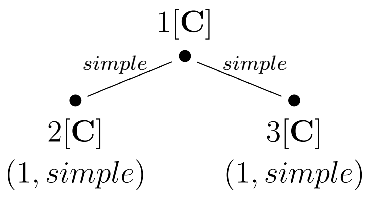
(i) f is preserving the vertex labels in `decoGraph2graph`(G_1):
$\forall v \in V_1 : f(v) \in V_2$ and $vertex\_name_{G_2}(f(v)) = vertex\_name_{G_1}(v)$,

(ii) f is preserving the `connected` relation and edge labels:
$\forall u, v \in V_1, \texttt{connected}_{G_1}(u,v) \Rightarrow \texttt{connected}_{G_2}(f(u), f(v))$ and $label_{G_1}(u,v) = label_{G_2}(f(u), f(v))$.

The image of G_1 by the morphism f is a subgraph of G_2 and is denoted by $f(G_1) = (V_2, f(E_1), f(lab_1))$:

- $f(E_1) = \{\{f(u), f(v)\} \mid \texttt{connected}_{G_1}(u,v) \text{ and } \{f(u), f(v)\} \in E_2\}$,
- $H(f(G_1)) = \{\{f(u), f(v)\} \mid \texttt{connected}_{G_1}(u,v) \text{ and } \{f(u), f(v)\} \in H_2\}$.

Example 3.10 If G_1 is the graph  and G_2 is



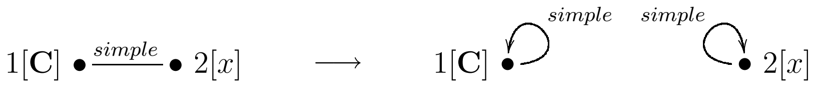
then $f_1 = \{1 \mapsto 1, 2 \mapsto 2\}$ and $f_2 = \{1 \mapsto 2, 2 \mapsto 3\}$ are two of the six morphisms from G_1 to G_2.

Definition 3.11 [Decorated labeled graph rewriting relation] Let $\mathcal{RG}$ be a decorated labeled graph rewriting system in $\mathcal{G}_W(\mathcal{L}, \mathcal{X})$. The decorated labeled graph *rewriting relation* associated to the rewriting system $\mathcal{RG}$ on $\mathcal{G}_V(\mathcal{L})$ is denoted $\longrightarrow_{\mathcal{RG}}$ and is defined by: for every decorated labeled graphs $G_1, G_2 \in \mathcal{G}_V(\mathcal{L})$, the decorated labeled graph G_1 rewrites to G_2, denoted $G_1 \longrightarrow_{\mathcal{RG}} G_2$, if there exists:

- a rewriting rule for decorated labeled graphs $g_l \to g_r \in \mathcal{RG}$, where $g_l, g_r \in \mathcal{G}_W(\mathcal{L}, \mathcal{X})$,
- a label substitution σ, and
- an injective morphism f from σg_l to G_1, such that $G_1 = f(\sigma g_l) \odot G_1{}'$;

and $G_2 = f(\sigma g_r) \odot G_1{}'$.
When the applied rule needs to be specified, the rewriting step is denoted by $G_1 \longrightarrow_{g_l \to g_r} G_2$.

Example 3.12 The rewriting step from Figure 8 is obtained by applying the rewriting rule



the label substitution $\{x \mapsto \mathbf{C}\}$, the morphism $f_1 = \{1 \mapsto 1, 2 \mapsto 2\}$, from example 3.10, and the decomposition given in Figure 6.

The rewriting step from Figure 9 is obtained by applying the same rewriting rule, the same label substitution, the morphism $f_2 = \{1 \mapsto 2, 2 \mapsto 3\}$, from example 3.10, and the decomposition given in Figure 7.

Let us remark that the set of decorated labeled graphs $\mathcal{G}_V(\mathcal{L})$ together with the rewriting relation $\longrightarrow_{\mathcal{RG}}$ is an abstract rewrite system and can thus enjoy properties of termination and confluence as described in [2,17].

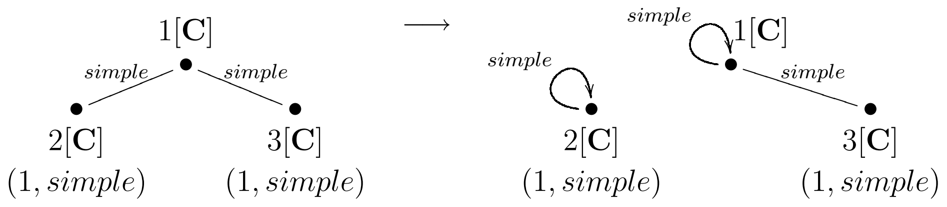


Fig. 8.

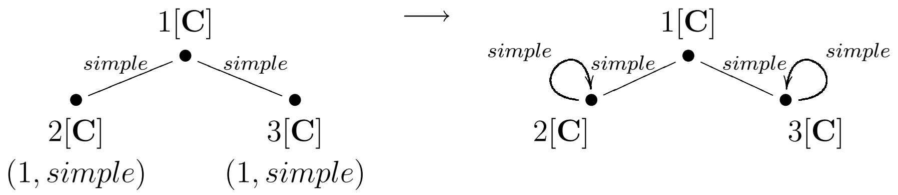


Fig. 9.

3.3 Rewriting minimally hidden forests

Rewriting decorated labeled graphs as labeled graphs does not guarantee the property that a (minimally hidden) forest is rewritten into a (minimally hidden) forest. Indeed this property is required if we want to implement forests as a multiset of terms and keep this structure by rewriting at the term level.

In order to get this property, we first transform the decorated labeled graph rewriting system $\mathcal{RG}$ into a decorated labeled forest $\mathcal{RF}$ such that for each rule $f_l \to f_r$ of $\mathcal{RF}$, f_l is a forest and f_r has only implicit edges. Transforming each rule $g_l \to g_r$ in $\mathcal{RG}$ in such a way is always possible, by using the hide-an-edge operation. Moreover $g_l = \texttt{forest2graph}(f_l)$ and $g_r = \texttt{forest2graph}(f_r)$.

Applying such a rule $f_l \to f_r \in \mathcal{RF}$ will never create new cycles but may produce a forest which is not a minimally hidden forest.

Then we can prove that a minimally hidden forest $F_{1,min}$ is rewritten using $\mathcal{RF}$ into a forest F_2, that can be transformed into a minimally hidden forest using the `merge` transformation, in order to simulate graph rewriting.

Theorem 3.13 (Soundness) *If $F_1 \longrightarrow_{\mathcal{RF}} F_2$ using the rewriting rule $f_l \to f_r$ then* $\texttt{forest2graph}(F_1) \to_{\mathcal{RG}} \texttt{forest2graph}(F_2)$ *using $g_l \to g_r$. If $F_2 \mapsto_{\texttt{merge}} F_{2,min}$ then* $\texttt{forest2graph}(F_2) = \texttt{forest2graph}(F_{2,min})$.

Theorem 3.14 (Completeness) *Let $F_{1,min}$ be a minimally hidden forest coding a labeled graph G_1 (i.e.* $\texttt{forest2graph}(F_{1,min}) = G_1$*). If $G_1 \longrightarrow_{\mathcal{RG}} G_2$ using the rewriting rule $g_l \to g_r$, then*

- *$F_{1,min} \longrightarrow_{\mathcal{RF}} F_2$ using the transformed rewriting rule $f_l \to f_r$,*
- *$F_2 \mapsto_{\texttt{merge}} F_{2,min}$,*

• $\texttt{forest2graph}(F_{2,min}) = \texttt{forest2graph}(F_2) = G_2$.

The result of the previous theorem is represented as follows.

$$\begin{tikzcd}
G_1 \arrow[r, "g_l \to g_r"'] & G_2 & & \text{labeled graph} \\
F_{1,min} \arrow[u, "forest2graph"] \arrow[r, "f_l \to f_r"'] & F_2 \arrow[u, "forest2graph"'] \arrow[r, phantom, "\mapsto \texttt{merge}"] & F_{2,min} \arrow[ul, bend right, "forest2graph"'] & \text{forest}
\end{tikzcd}$$

Proof. The graphs f_l, g_l on one hand, f_r, g_r on the other hand, have the same connected vertices by construction. The morphism used to apply $g_l \to g_r$ on G_1 can be transposed to a decorated labeled graph morphism used to apply $f_l \to f_r$ on $F_{1,min}$ to get F_2 and $\texttt{forest2graph}(F_2) = G_2$. Then applying the rule `merge` if needed, will only reveal some edges but will not change connected vertices and will not introduce cycle. So $F_{2,min}$ is a minimally hidden forest and $\texttt{forest2graph}(F_{2,min}) = G_2$. □

4 Rewriting decorated terms

We address in this section the transformation of a minimally hidden forest into a multiset of terms. Each connected component of the minimally hidden forest is a tree (a connected acyclic graph in graph theory), that can be encoded by a term structure. The multiset of terms is coded using the associative-commutative operator $+$.

4.1 Decorated terms

A term can be considered as a tree with a distinguished vertex, that corresponds to the root of the term, plus some order on its children, corresponding to its subterms, plus some order on the children of its children, corresponding to the sub-sub-terms and so on. That is, a term can be obtained from a tree if we choose a root and an order on vertices. If we have associative commutative (AC) matching, we may assume that the list of subterms of a term is built with an associative and commutative operator, so that two terms that differ only by the order on vertices are equal modulo AC. In what follows, we assume this is the case. Otherwise, one just needs in the following discussion to associate to a rule r not only a rule r', but several rules r' according to the possible orders.

Definition 4.1 [Decorated term] A *decorated term* is a decorated labeled tree with a root (that is with a marked vertex $r \in V$). A decorated term is also called a *vision* of the decorated labeled tree.

A tree with m edges has $n = m + 1$ vertices. Hence, a decorated labeled tree with m edges has $m+1$ visions. The number of visions corresponding to a connected graph is bounded by $O(m2^m)$, i.e. $O(2^{O(m)})$. If we consider the class of connected graphs whose cyclomatic number κ is bounded by some constant κ_0, the number of visions of a connected graph with m edges is bounded by a polynomial in n (in $O(m^{\kappa_0+1})$).

Definition 4.2 [`forest2terms` and `terms2forest`] Given a forest F, with k connected components and given a root for each connected component, we call `forest2terms` the transformation that maps F to $t_1 + t_2 + \ldots + t_k$, where each t_i is a decorated term associated to the ith connected component. We call `terms2forest` the reverse operation (i.e., we forget the roots).

Given an order on the k connected components and given hidden edges in each connected component, if F has m edges, then there are $m+k$ such terms obtained by choosing the roots of the k connected components.

Definition 4.3 [Equivalent decorated terms] Two decorated terms t_1 and t_2 are equivalent if the underlying decorated labeled trees are equivalent by definition 2.9. We denote this equivalence by $t_1 \sim t_2$.

Let us denote by $+$ the multiset constructor that is associative and commutative. The equivalence relation $\sim$ on decorated terms is extended to multisets as follows: $t_1 + t_2 + \cdots + t_k \sim u_1 + u_2 + \cdots + u_k$ if there exists a permutation π on $\{1, \cdots, k\}$ and $t_i \sim u_{\pi(i)}$.

The set of all visions of a decorated term is the set of all equivalence classes:

Definition 4.4 [All visions of a decorated term] $\texttt{AllVision}(t) = [t]_\sim$

From the discussion above, the equivalence relation class of a given decorated term t_i with m_i edges has at most $2^{O(m_i)}$ elements in the worst case. The equivalence class of t has at most $2^{O(m_1)}2^{O(m_2)} \ldots 2^{O(m_k)} = 2^{O(m)}$ elements. If the cyclomatic number κ of the graph `forest2graph(terms2forest(t))` is bounded by some constant, the equivalence relation class of t has at most a polynomial number of elements. Each class modulo $\sim$ has in turn $k!$ elements.

4.2 Rewriting relation for decorated terms

Now, we show how to simulate a forest rewriting step with rewriting in equivalence classes.

We first need to transform the decorated labeled forest rewriting system $\mathcal{RF}$ into a term rewriting system. From previous constructions, we may assume that the rules $r : f_l \to f_r \in \mathcal{RF}$ are such that f_r has only implicit edges. Let $t_1, \ldots, t_k$ be the connected components of f_l. Let $r_i : f_{l,i} \to f_{r,i}$ be the rewriting rules on graphs such that $f_{l,i} = t_i$ and $f_{r,i}$ is the subgraph of f_r restricted to the vertices of t_i ($f_{r,i}$ can be not well-formed).

Proposition 4.5 *Let T, T' be decorated labeled trees. If T rewrites to T' using r, then there exist $T_1, \dots, T_k$ such that $T \rightarrow_{r_1} T_1 \rightarrow_{r_2} T_2 \rightarrow \cdots \rightarrow_{r_k} T_k = T'$.*

Proposition 4.6 *If $T \rightarrow_{r_1} T_1 \rightarrow_{r_2} T_2 \rightarrow \cdots \rightarrow_{r_k} T_k = T'$ and the images by the morphisms used in the k rules do not overlap, then T rewrites to T' using r.*

As a consequence, we may restrict to rules of the form $r : f_l \rightarrow f_r$ where f_l is a tree (and applying the rules in some order, if needed). We encode this rule r by a rewriting rule on terms obtained as follows:

$$r' : t + X \rightarrow u_1 + u_2 + \cdots + u_p + X$$

where t is a decorated term representing the tree f_l (obtained by choosing a root), X is a variable, and $u_1, u_2, \dots, u_p$ are the decorated terms coding the vertices of f_r (recall that all edges are implicit in f_r).

Theorem 4.7 (Soundness) *If $t_1 + t_2 + \cdots + t_k \rightarrow s_1 + s_2 + \cdots + s_{l'}$ using r' then* $\texttt{terms2forest}(t_1 + t_2 + \cdots + t_k) \rightarrow \texttt{terms2forest}(s_1 + s_2 + \cdots + s_{l'})$ *using r.*

Proof. Let T_i be $\texttt{terms2forest}(t_i)$ for $i = 1, \dots, k$ and $T = \texttt{terms2forest}(t_1 + t_2 + \cdots + t_k)$. If rule r' is applied, then t matches some t_i, and X is $t_1 + \cdots + t_{i-1} + t_{i+1} \cdots + t_k$; so $s_1 + s_2 + \cdots + s_{l'}$ is $t_1 + \cdots + t_{i-1} + t_{i+1} \cdots + t_k + v_1 + v_2 + \cdots + v_p$, where the v_j's for $j = 1, \dots, p$, are the images of u_j's with the matching morphism. Assume without lost of generality that $i = 1$. Since t matches t_1, that means that t_1 and t have the same root. So there exists a morphism that maps $f_l = \texttt{terms2forest(t)}$ into $\texttt{terms2forest(t}_1\texttt{)}$; r can be applied on T and T rewrites to the graph where the `connected` relation is unchanged on $T_2, \dots T_k$, and changed according to r' on T_1. $\texttt{terms2forest}(s_1 + s_2 + \cdots + s_{l'})$ is this graph. □

Theorem 4.8 (Completeness) *If $F_1 \longrightarrow_{\mathcal{RF}} F_2$ using some rule $r : f_l \rightarrow f_r$, where f_l is a tree, then there exists a decorated term $t_1 + t_2 + \cdots + t_k$ with*

- $\texttt{terms2forest}(t_1 + t_2 + \cdots + t_k) = F_1$,
- *$t_1 + t_2 + \cdots + t_k \rightarrow s_1 + s_2 + \cdots + s_{l'}$ using the rewriting rule r', and*
- $\texttt{terms2forest}(s_1 + s_2 + \cdots + s_{l'}) = F_2$.

Proof. If $F_1 \longrightarrow F_2$ using r, then there is a morphism from f_l to F_1. Let be $T_1, \dots, T_k$ the k connected components of F_1. Since f_l is connected, this morphism sends f_l to some T_i. Assume without lost of generality that $i = 1$.

$T_2, \dots, T_k$ are unchanged by r. Consider t_i as any decorated term coding T_i, for $i > 1$. Consider t_1 as the decorated term coding T_1, where the root in t_1 is the root of t in rule r', and with the same set of hidden edges as in t. We have `terms2forest`$(t_1 + t_2 + \dots + t_k) = F_1$, $t_1 + t_2 + \dots + t_k \to v_1 + v_2 + \dots + v_p + t_2 + \dots + t_k$ using r', and `terms2forest`$(v_1 + v_2 + \dots + v_p + t_2 + \dots + t_k) = F_2$. □

Given $t_1 + t_2 + \dots + t_k$ with `terms2forest`$(t_1 + t_2 + \dots + t_k) = F_1$, one may need to find a suitable equivalent vision $u_1 + u_2 + \dots + u_k$ on which the rule r' can be applied. This gives the following diagram:

$$\begin{tikzcd}
 & & F_1 \arrow[r, "r:f_l \to f_r"'] & F_2 \\
t_1 + t_2 + \dots + t_k \arrow[urr, bend left, "terms2forest"] & \sim & u_1 + u_2 + \dots + u_k \arrow[u, "terms2forest"] \arrow[r, "r'"'] & s_1 + s_2 + \dots + s_{l'} \arrow[u, "terms2forest"']
\end{tikzcd}$$

In this transformation, the rewrite rule r is realized by a unique rule r'. However, to apply the rule r', one must find some $u_1 + u_2 + \dots + u_k$ in $AllVision(t_1 + t_2 + \dots + t_k)$, that may possibly contain $2^{O(m)}$ elements, where m is the number of edges of F_1 (and a polynomial number if the cyclomatic number of F_1 is bounded).

If we have some rewriting rules on terms that map any term representation of a forest into its visions, denoted by $\vdash^{vision}$, and some rules that implement `merge` transformation on term representation of forests, denotes by $\vdash^{merge}$ we get the following diagram:

$$\begin{tikzcd}
 & G_1 \arrow[r, "g_l \to g_r"'] & G_2 & & \\
 & F_{1,min} \arrow[u, "forest2graph"] \arrow[r, "f_l \to f_r"'] & F_2 \arrow[u, "forest2graph"] & \mapsto_{\texttt{merge}} & F_{2,min} \arrow[ull, "forest2graph"'] \\
t_1 + \dots + t_k \vdash^{vision} \arrow[ur, bend left, "terms2forest"] & u_1 + \dots + u_k \arrow[u, "terms2forest"] \arrow[r, "r'"'] & s_1 + \dots + s_{l'} \arrow[u, "terms2forest"'] & \vdash^{merge} & w_1 + \dots + w_n \arrow[u, "terms2forest"]
\end{tikzcd}$$

The merge operation can be realized using classical algorithms (see [8]) in polynomial time. Moreover, from above discussions $\vdash^{vision}$ may relate a term to $2^{O(m)}$ terms modulo associativity and commutativity of $+$. Hence, one simulation step going from $t_1 + \dots + t_k$ to $w_1 + \dots + w_n$, may require to test the matching of the left-hand side of rule r' to $2^{O(m)}$ terms in the worst case. This has to be balanced by the fact that rewriting graphs implies solving graph isomorphism problems, for which no polynomial bound is known.

In our context, and from our experience in the GasEl system, we are far from this worst case analysis. This is due to various facts:

- Associative and commutative matching is simply exponential in general, but can be efficiently performed by current AC-compilers.
- Furthermore, due to their structure, rewrite rules r' only apply on top of terms $u_1 + \cdots + u_k$ which improves efficiency of the AC-matching problems solving.
- From our experience, the involved rules in our application related to chemistry imply only molecules with few cycles. As a consequence, in practice our approach is rather good for chemical rules.
- In some sense, if the number of cycles involved is low, we are close to term rewriting. An implementation with adjacency lists, would not so directly benefit from this property.

5 Implementation

The implementation of the GasEl system relies on the concepts presented in this paper. It has been designed in the ELAN language that provides in particular an efficient rewriting engine for associative and commutative theories [20]. In [5,15], the basic concepts of the chemical kinetics, together with the chemical and computational problems related to the conception and validation of a reaction mechanism are presented. A general structure for the generator of reaction mechanisms called GasEl is described in [15].

In GasEl the relations `graph2forest` and `forest2graph` are performed by the user interface, based on SMILES editor and viewer [23,1]; the relations `forest2tree` and `tree2forest` are performed by the ELAN compiler, with a suitable definition of GasEl terms. The `vision` and `merge` relations are in fact the key of simulating graph rewriting by term rewriting. Intuitively, generating all visions of a term is performed by selecting successively every node as root, which amounts to visit the tree [8]. The merge operation may be very complex in general. It has been fully implemented in the context of the chemical study of ten generic reactions of the oxidizing pyrolysis.

The prototype makes an extensive use of associative-commutative rewriting and of the ELAN strategy language which appeared as perfectly suitable for expressing the control of the chemical reactions chaining occurring in the automated generation of reaction mechanisms.

Qualitative chemical validations of the prototype show that our approach gives, for acyclic molecules, the same results as the existing mechanism generators, and for polycyclic molecules produces validated original results.

The main concerns of the GasEl project ware to be able to rewrite molecular graphs (with cycles), to explore the computational foundation of the automated generation of reaction mechanisms, and to provide a flexible and reliable tool that gives significant insights to the chemists. Significant progresses have been achieved in this directions, that need to be confronted to other chemical computational systems. However the comparison with other systems solving the same chemical problem is very difficult since the code of concurrent systems is mostly confidential and often not distributed.

6 Conclusion

In this paper we explore a class of graphs and a graph rewriting relation where vertices are preserved and only edges are changed. We show how to represent cyclic labeled graphs by decorated labeled trees or forests, then how to transform trees into terms. A graph rewriting relation is defined, then simulated by a forest rewriting relation, which can be in turn simulated by a rewriting relation on equivalence classes of terms. As a consequence, this kind of graph rewriting can be implemented using term rewriting which is a good level of formalization. This study is motivated by the design of the GasEl system for the generation of kinetics reactions mechanisms.

As further work, it would be interesting to compare this term-based implementation of graph rewriting with a direct implementation of graph rewriting in a system able to take benefit from the specific class of graphs we are considering. Another interesting question is to find the right level of formalism to study properties like termination, confluence, sufficient completeness,... of graph rewriting rules in this class of graphs.

Finally, we feel that our approach of graph rewriting is worth applying in some others contexts such as networks or web services. This had to be investigated.

References

[1] Marvin: A tool for Molecule Drawing and Visualization. http://www.chemaxon.com/marvin/.

[2] Franz Baader and Tobias Nipkow. *Term Rewriting and all That.* Cambridge University Press, 1998.

[3] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewriting. *Math. Systems Theory*, 20:83–127, 1987.

[4] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An Overview of ELAN. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the Second International Workshop on Rewriting Logic and Applications*, volume 15, http://www.elsevier.nl/locate/entcs/volume15.html, Pont-à-Mousson (France), September 1998. Electronic Notes in Theoretical Computer Science. Rapport LORIA 98-R-316.

[5] Olivier Bournez, Guy-Marie Côme, Valérie Conraud, Hélène Kirchner, and Liliana Ibănescu. A Rule-Based Approach for Automated Generation of Kinetic Chemical Mechanisms. In Robert Nieuwenhuis, editor, *Proceedings of the 14th International Conference on Rewriting Techniques and Applications, RTA 2003, Valencia, Spain, June 9-11, 2003*, volume 2706 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2003.

[6] Olivier Bournez, Guy-Marie Côme, Valérie Conraud, Hélène Kirchner, and Liliana Ibănescu. Automated Generation of Kinetic Chemical Mechanisms Using Rewriting. In P.M.A. Sloot, D. Abramson, A.V. Bogdanov, J.J. Dongarra, A.Y. Zomaya, and Y.E. Gorbachev, editors, *Proceedings of the International Conference on Computational Science, ICCS 2003, Melbourne, Australia, June 2-4, 2003, Part III*, volume 2659 of *Lecture Notes in Computer Science*, pages 367–376. Springer, 2003.

[7] Guy-Marie Côme. *Gas-Phase Thermal Reactions. Chemical Engineering Kinetics.* Kluwer Academic Publishers, 2001.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill Book Company, second edition, 2001.

[9] Andrea Corradini and Ugo Montanari. An algebra of graphs and graph rewriting. In David H. Pitt, Pierre-Louis Curien, Samson Abramsky, Andrew M. Pitts, Axel Poigné, and David E. Rydeheard, editors, *Category Theory and Computer Science*, volume 530 of *Lecture Notes in Computer Science*, pages 236–260. Springer, 1991.

[10] Andrea Corradini and Francesca Rossi. Hyperedge replacement jungle rewriting for term-rewriting systems and logic programming. *Theor. Comput. Sci.*, 109(1-2):7–48, 1993.

[11] Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. Hyperedge replacement, graph grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars*, pages 95–162. World Scientific, 1997.

[12] James Dugundji and Ivar Ugi. An Algebraic Model of Constitutional Chemistry as a Basis for Chemical Computer Programs. *Topics in Current Chemistry*, 39:19–64, 1973.

[13] P. Fradet and D. Le Métayer. Structured gamma. *Science of Computer Programming*, 31(2-3):263–289, 1998.

[14] Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science.* Springer, 1992.

[15] Liliana Ibanescu. *Programmation par règles et stratégies pour la génération automatique de mécanismes de combustion d'hydrocarbures polycycliques.* Thèse de Doctorat d'Université, Institut National Polytechnique de Lorraine, Nancy, France, June 2004. `http://www.loria.fr/~ibanescu/these_en.html`.

[16] Hélène Kirchner and Pierre-Etienne Moreau. Promoting rewriting to a programming language: A compiler for non-deterministic rewrite programs in associative-commutative theories. *Journal of Functional Programming*, 11(2):207–251, 2001.

[17] J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter 6. Oxford University Press, 1990.

[18] Patrick Lincoln, Narciso Martí-Oliet, and José Meseguer. Specification, transformation, and programming of concurrent systems in rewriting logic. In G. Blelloch, K. Chandy, and S. Jagannathan, editors, *Specification of Parallel Algorithms*, volume 18 of *DIMACS Series*, pages 309–339. American Mathematical Society, 1994.

[19] José Meseguer. Research directions in rewriting logic. In *Computational Logic*, volume 165 of *Lecture Notes in Computer Science*, Marktoberdorf, Germany, 1997. NATO Advanced Study Institute, Springer-Verlag.

[20] Pierre-Etienne Moreau and Hélène Kirchner. A Compiler for Rewrite Programs in Associative-Commutative Theories. In Catuscia Palamidessi, Hugh Glaser, and Karl Meinke, editors, *Principles of Declarative Programming, 10th International Symposium, PLILP'98 Held Jointly with the 7th International Conference, ALP'98, Pisa, Italy, September 16-18, 1998, Proceedings*, volume 1490 of *Lecture Notes in Computer Science*, pages 230–249. Springer, 1998.

[21] Jean-Claude Raoult and Frédéric Voisin. Set-Theoretic Graph Rewriting. In *Proceedings of the International Workshop on Graph Transformations in Computer Science*, Lecture Notes in Computer Science, pages 312–325, 1993.

[22] Alison S. Tomlin, Tamás Turányi, and Michael J. Pilling. *Mathematical Tools for the Construction, Investigation and Reduction of Combustion Mechanisms*, volume 35 of *Comprehensive Chemical Kinetics*, chapter 4, pages 293–437. Elsevier, Amsterdam, 1997.

[23] David Weininger. SMILES — A Language for Molecules and Reactions. In Johann Gasteiger, editor, *Handbook of Chemoinformatics. From Data to Knowledge*, pages 80–102. Wiley-VCH, 2003.