

Hyperedge replacement jungle rewriting for term-rewriting systems and logic programming*

Andrea Corradini and Francesca Rossi

Università di Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy.

Abstract

Corradini, A. and F. Rossi, Hyperedge replacement jungle rewriting for term-rewriting systems and logic programming, *Theoretical Computer Science* 109 (1993) 7–48.

We introduce hyperedge replacement jungle rewriting, a graph-rewriting formalism suitable for modeling the manipulation of terms and similar structures, and investigate its expressive power by showing that it can model both term-rewriting systems and logic programming in a faithful way. For term-rewriting systems we prove the soundness of their jungle representation, and a result of completeness w.r.t. applicability which is stronger than similar results in the related literature, since it works also for non-left-linear rules. For logic programming both soundness and completeness hold.

0. Introduction

In this paper we investigate the properties of *hyperedge replacement jungle rewriting*, a formalism based on graph rewriting, and demonstrate its expressive power, showing that it can model faithfully both term-rewriting systems (TRSs) and logic programming.

Jungles are special acyclic directed hypergraphs (i.e., graphs where each edge can have any number of source and target nodes), and have been introduced in [29, 20] to represent sets of terms with possibly shared subterms. In this sense they can be thought of as being the hypergraph corresponding to the well-known *directed acyclic graphs (dags)* (see [30]).

Correspondence to: A. Corradini, Università di Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy. Email addresses of the authors: andrea@di.unipi.it and rossi@di.unipi.it.

* Research partially supported by the GRAGRA Basic Research Esprit Working Group no. 3299.

The *theory of graph rewriting* (or *graph grammars*) basically studies a variety of formalisms which extend the theory of formal languages in order to deal with structures more general than strings, like graphs and maps. A graph-rewriting system is a collection of graph rewrite rules; each rule can be applied to a graph to replace an occurrence of its left-hand side with its right-hand side. The form of graph rewrite rules and the mechanisms stating how a rewrite rule can be applied to a graph and what the resulting graph is, depend on the specific formalism. Among the various formulations of graph rewriting the so-called algebraic approach [16, 11] is one of the most successful, mainly because of its flexibility. In fact, since the basic notions of rule and rewriting step are defined in terms of diagrams and constructions in a category, they can be defined in a uniform way for a wide range of structures simply by changing the underlying category [12]. Moreover, many results can be proved once and for all using categorical techniques.

Jungle rewriting is defined in this paper by applying the guidelines of the algebraic approach to graph rewriting to the category of jungles. The resulting formalism is only apparently similar to “jungle evaluation” as defined in [20–22], which is actually the result of applying the algebraic approach to the category of hypergraphs, and then by restricting the attention to those hypergraphs which are jungles.

A *hyperedge replacement* jungle rewrite rule is a rule such that the left-hand side is a jungle consisting of a single hyperedge (and of the nodes connected to it). Graph-rewriting systems satisfying a similar restriction have been studied in [3, 18], but both consider arbitrary hypergraphs.

In summary, hyperedge replacement jungle rewriting can be defined easily as a mixture of well-known notions. Despite that, its expressive power as a rewriting formalism has never been explored before. The main achievement of this paper is that both term-rewriting systems and logic programming can be modeled by hyperedge replacement jungle rewriting, and that for TRSs the proposed representation enjoys better properties than all the similar proposals we are aware of. A preliminary version of this paper is [6], where just the relationship between TRSs and jungle rewriting has been considered.

Many papers in the literature investigate *term graph rewriting*, i.e., the issue of representing TRSs using graph rewriting, either by exploiting the algebraic theory of graph grammars (like [28, 22] that follow the so-called double-pushout approach, and [30, 23, 24] that use instead a single pushout construction), or by using a more operational presentation of graph rewriting (e.g., [2]). In general, term rewrite rules are translated into graph rewrite rules, while terms are represented either by dags or by jungles: usually, the same term can be represented by many graphs which are not isomorphic. All the mentioned papers prove the soundness of their graph representation of rewrite rules, in the sense that if a graph rule which represents a term rule can be applied to a graph, then the corresponding term rule can be applied to the term represented by the graph. However, the translation is not complete w.r.t. applicability, i.e., it is possible that a graph cannot be rewritten with a graph rule, although the term it represents can be reduced by the corresponding term rewrite rule. Usually, this kind

of completeness just holds for left-linear rules, i.e., rules which do not have two occurrences of the same variable on the left-hand side. Some of those papers propose additional mechanisms which allow one to deal with non-left-linear rules, like *folding rules* as in [20–22], or a generalization of the notion of *occurrence* as in [23].

On the contrary, the jungle representation that we propose for TRS enjoys completeness w.r.t. applicability for non-left-linear rules as well, without resorting to additional mechanisms. This is a consequence of the basic choice of performing the constructions which implement the rewriting (i.e., the pushout complement, along the guidelines of the algebraic approach) directly in the category of jungles. In fact, the pushout in such a category corresponds to term unification, which is obviously powerful enough to model the pattern-matching mechanism needed to perform term rewriting. It could be argued that resorting to unification would unnecessarily increase the complexity of term rewriting, but, although we do not make a full complexity analysis, we give some evidence that in the case of left-linear rules and ground rewriting the cost of unification should be comparable to that of pattern matching.

The possibility of modeling full term unification in our framework allows us to represent logic programming as well, which has never been considered in previous works on term graph rewriting. The representation of clauses as jungle rewrite rules, of resolution steps as direct derivations, of logic programs as hyperedge replacement jungle grammars, and the corresponding results of soundness and completeness, are essentially a cleaner rephrasing of similar results presented in a previous paper [5]. In that paper, however, the emphasis was placed on proposing an original graph representation of logic programs, rather than on analyzing the expressive power of a rewriting formalism. For example, terms and formulas were represented in an ad hoc way as a mixture of dags and jungles, and some proofs were easier, thanks to the particular representation.

We must stress that the main goal of this paper is not to provide new results for the theory of term-rewriting systems or of logic programming, but instead that of investigating the expressive power of a new graph-rewriting formalism. Nevertheless, we believe that the fact that both term-rewriting systems and logic programming can be represented uniformly within the same formalism is very promising and worth investigating. The first idea which comes into mind is, of course, the possibility of a framework where both paradigms are present, and can be used together or separately without any need of switching context from one to the other. Along these lines, we could, for example, envision a framework where logic programs are considered modulo an equational theory, which is represented by a term-rewriting system. Also *conditional* TRSs seem to be easily representable in our formal framework.

Finally, a remarkable property of hyperedge jungle rewriting systems is that, under very reasonable hypotheses, a term-rewriting system or a logic program can be extracted from such systems. This fact suggests that the expressive power of hyperedge replacement jungle rewriting should be essentially equivalent to some kind of term rewriting with unification, thus making this formalism interesting on its own, and not

a mere graphical model into which one can translate other formalisms. This topic deserves further investigation.

The paper is organized as follows. Section 1 introduces the category of jungles, describes how jungles can be used to represent sets of terms, and studies the conditions for the existence, in that category, of pushouts and pushout complements, i.e., the basic categorical constructions needed to apply the algebraic approach. Then Section 2 presents the definition of hyperedge replacement jungle rewriting as the application of the algebraic approach to graph rewriting to the category of jungles, with the additional restriction of replacing exactly one edge at each step: the effects of this restriction are compared with the hyperedge replacement hypergraph-rewriting formalisms proposed in [3, 18]. Section 3 presents our proposal of term graph rewriting, showing that a term-rewriting system can be represented by a hyperedge replacement jungle-rewriting system in a sound way, which is complete w.r.t. applicability also for non-left-linear rules. Then Section 4 shows that the same framework, when applied to a logic programming signature, can represent logic programs in a sound and complete way. Section 5 discusses the relationship between our proposal and other related works in the literature, mainly with respect to the handling of non-left-linear rules. Finally, Section 6 concludes the paper by summarizing its results and giving some hints for future developments.

1. The category of jungles and its properties

In this section we introduce the category of jungles, describe how jungles can be used to represent terms with possibly shared subterms, and explore some properties of the category (i.e., existence of pushouts and pushout complements) which are relevant for the definition of jungle rewriting along the guidelines of the algebraic approach to graph grammars. Jungles and jungle rewriting (also called “jungle evaluation”) have been introduced in [29, 20–22] to model term-rewriting systems via graph rewriting, thus providing a rich contribution to the area of graph term rewriting. Most of the definitions and results of Sections 1.1 and 1.2 are borrowed from the works by Hoffmann and Plump [21, 22]. The propositions about existence of pushouts and pushout complements presented in Section 1.3 generalize similar results presented in [5].

1.1. The categories of hypergraphs and jungles

A (directed) hypergraph straightforwardly generalizes a (directed) graph: it includes a set of nodes and a set of hyperedges. Every hyperedge has a list of source nodes and a list of target nodes, instead of exactly one source and one target node. Nodes and hyperedges are colored by elements of color sets. In this paper we are interested just in hypergraphs whose nodes (hyperedges) are labeled by the sorts (operators) of a many-sorted signature.

Definition 1.1 (*Many-sorted signature*). A many-sorted signature $\underline{\Sigma}$ is a pair $\underline{\Sigma} = (S, \Sigma)$, where S is a set of sorts and $\Sigma = \{\Sigma_{w,s}\}$ is a family of sets of operators indexed by $S^* \times S$. If $f \in \Sigma_{w,s}$ then s is called the *type* of f , w is its *arity*, and $|w|$ (the length of w) is the *rank* of f .

Definition 1.2 (*Hypergraphs*). A hypergraph G over $\underline{\Sigma}$ is a tuple $G = (V, E, s, t, m, l)$, where V is a set of nodes (or vertices), E is a set of hyperedges, s and $t: E \rightarrow V^*$ are the source and target functions, $l: V \rightarrow S$ maps each node to a sort of $\underline{\Sigma}$, and $m: E \rightarrow \Sigma$ maps each edge to an operator of $\underline{\Sigma}$.

A path in a hypergraph from node v to node u is a finite sequence of triples $\langle \langle k_1, e_1, j_1 \rangle, \dots, \langle k_n, e_n, j_n \rangle \rangle$, where all k_i, j_i are natural numbers, all e_i are hyperedges, and such that $s(e_1)|_{k_1} = v$, $t(e_n)|_{j_n} = u$, and for all $1 \leq i < n$, $t(e_i)|_{j_i} = s(e_{i+1})|_{k_{i+1}}$ (here $s|_i$ denotes the i th element of the list of nodes s). By convention, there is exactly one empty path “ $\langle \quad \rangle$ ” from each node to itself. A hypergraph is *acyclic* if there are no nonempty paths from one node to itself.

A hypergraph is *discrete* if it contains no hyperedges. For a hypergraph G , $\text{indegree}_G(v)$ ($\text{outdegree}_G(v)$) denotes the number of occurrences of a node v in the target strings (source strings) of all hyperedges of G .

For the sake of simplicity, in the rest of the paper we will often omit the prefix “hyper-”, calling hypergraphs and hyperedges simply graphs and edges. A *jungle* is a hypergraph satisfying some additional conditions which makes it suitable to represent sets of (finite) terms with possibly shared subterms. The precise role played by these restrictions will be made clear in Section 1.2.

Definition 1.3 (*Jungles*). A jungle over $\underline{\Sigma}$ is a graph $G = (V, E, s, t, m, l)$ over $\underline{\Sigma}$ such that

- the labeling of the edges is consistent with both the number and the labeling of the connected nodes, i.e., $\forall e \in E, m(e) \in \Sigma_{w,r} \Leftrightarrow l^*(s(e)) = r$ and $l^*(t(e)) = w$, where l^* is the obvious extension of l to tuples;
- for each node $v \in V$, $\text{outdegree}_G(v) \leq 1$;
- G is acyclic.

Example 1.4 (*Graphical representation of hypergraphs and jungles*). Let $\underline{\Sigma} = (S = \{\text{list}, \text{nat}\}, \Sigma = \{\Sigma_{\varepsilon, \text{list}} = \{\text{EMPTY}\}, \Sigma_{\varepsilon, \text{nat}} = \{0\}, \Sigma_{\text{nat-list}, \text{list}} = \{\text{cons}, \text{rem}\}, \Sigma_{\text{nat}, \text{nat}} = \{\text{succ}\}, \Sigma_{\text{nat-nat}, \text{nat}} = \{+\}, \Sigma_{\text{list}, \text{nat}} = \{\text{car}\}\})$ (ε denotes the empty word of a free monoid). Then there are two hypergraphs over $\underline{\Sigma}$, called, respectively, G and H as shown in Fig. 1.

In the graphical representation of hypergraphs we will use the following conventions. Edges and nodes are represented by boxes and circles, respectively, with the colors written inside. The source (target) nodes of an edge are connected to the edge itself by an arrow (sometimes called a *tentacle*) oriented towards the box (circle). When an edge has more than one source (target), the incoming (outgoing) tentacles are

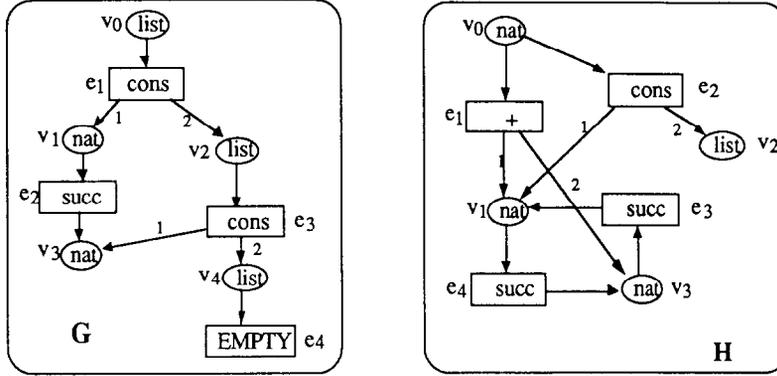


Fig. 1.

numbered. The graphical representation we use puts in evidence that hypergraphs can be regarded as directed bipartite graphs.

It is easy to check that G above is also a jungle over $\underline{\Sigma}$, while H is not. Actually, no condition of Definition 1.3 holds for H , since there is a node with outdegree 2 (v_0), the sort of the edge labeled by $cons$ is not consistent with the type, and there is a cycle.

Definition 1.5 (*Hypergraph morphisms*). A morphism of hypergraphs (over $\underline{\Sigma}$) $f: G_1 \rightarrow G_2$ consists of a pair of functions between edges and nodes, respectively, which are compatible with the source and target functions and are color-preserving. More precisely, if $G_1 = (V_1, E_1, s_1, t_1, m_1, l_1)$ and $G_2 = (V_2, E_2, s_2, t_2, m_2, l_2)$ are two graphs over $\underline{\Sigma}$, then $f = (f_V: V_1 \rightarrow V_2, f_E: E_1 \rightarrow E_2)$ such that $s_2 \circ f_E = f_V^* \circ s_1$; $t_2 \circ f_E = f_V^* \circ t_1$; $m_2 \circ f_E = m_1$; and $l_2 \circ f_E = l_1$, where f_V^* is the obvious extension of f_V to lists of nodes.

A hypergraph morphism $f = (f_V, f_E): G_1 \rightarrow G_2$ is *injective* (*surjective*) iff both f_V and f_E are injective (*surjective*) functions. A morphism f is an *isomorphism* if it is both injective and surjective.

Morphisms of jungles are defined exactly in the same way. Because of the peculiar structure of jungles, a jungle morphism is uniquely determined by its behavior on the “root nodes” [22], as formalized by the following fact.

Fact 1.6 (*Characterization of jungle morphisms*). If G is a jungle, let $ROOT_G = \{v \in V_G \mid indegree_G(v) = 0\}$ be the set of its roots. Then two morphisms $f, g: G \rightarrow H$ are identical if and only if $f_V(v) = g_V(v)$ for each $v \in ROOT_G$.

Definition 1.7 (*Categories $\mathbf{HGraph}_{\underline{\Sigma}}$ and $\mathbf{Jungle}_{\underline{\Sigma}}$*). The category whose objects are directed hypergraphs over $\underline{\Sigma}$ and whose arrows are hypergraph morphisms will be denoted by $\mathbf{HGraph}_{\underline{\Sigma}}$. The full subcategory of $\mathbf{HGraph}_{\underline{\Sigma}}$ including all jungles will be called $\mathbf{Jungle}_{\underline{\Sigma}}$.

1.2. Representing terms by jungles

As it should be evident by Example 1.4 there is a fairly obvious correspondence between the jungles over a signature $\underline{\Sigma}$ and the terms built over that signature. Although jungles have been widely used during the last few years by people interested in graph rewriting for term rewriting [29, 20–22, 24] or for logic programming [5, 7], one cannot disregard the other main graphical representation of terms with shared subterms, namely, *directed acyclic graphs* in the many variants, which have been used, among others, in [30, 28, 23, 3]. The relationship between the two representations has been made precise in [5], where it has been shown that the category of jungles and that of dags over the same signature are equivalent. Thus, as far as one is concerned with results or constructions expressible in categorical terms, the use of jungles or dags is completely interchangeable. Our preference for jungles is, therefore, mainly a matter of taste (but see also the observation at the end of Section 2.2).

We briefly recall here the formal correspondence between jungles and terms, and between jungle morphisms and term substitutions. Most of the definitions are borrowed from [21, 22]. In the rest of this section we refer to a fixed many-sorted signature $\underline{\Sigma} = (S, \Sigma)$.

Definition 1.8 (*Terms, substitutions, unifiers*). Let $X = \{X_s\}_{s \in S}$ be a family of pairwise disjoint sets of variables indexed by S . Then a *term of sort s* is an element of $T_{\underline{\Sigma}}(X)$ (the carrier of sort s of the free $\underline{\Sigma}$ -algebra generated by X), that is a variable of X_s , or a constant in $\Sigma_{c,s}$, or $f(t_1, \dots, t_n)$, if $f \in \Sigma_{w,s}$, $w = s_1 \dots s_n$, and t_i is a term of sort s_i for each $1 \leq i \leq n$. The sort of a term t will be denoted by $\text{sort}(t)$.

Given two families of S -indexed sets of variables $X = \{X_s\}$ and $Y = \{Y_s\}$, a *substitution σ from X to Y* is a function $\sigma: X \rightarrow T_{\underline{\Sigma}}(Y)$ that is sort-preserving, i.e., such that $\sigma(x) \in T_{\underline{\Sigma}}(Y)$ for each $x \in X_s$. When X is finite, σ can be represented as $\sigma = \{x_1/t_1, \dots, x_n/t_n\}$, where $\sigma(x_i) = t_i$. By σ_s we denote the restriction of σ to X_s .

By the freedom property of $T_{\underline{\Sigma}}(X)$, a substitution $\sigma: X \rightarrow T_{\underline{\Sigma}}(Y)$ can be extended in a unique way to a Σ -homomorphism $\sigma^\# : T_{\underline{\Sigma}}(X) \rightarrow T_{\underline{\Sigma}}(Y)$, defined as $\sigma^\#(x) = \sigma(x)$ if x is a variable in X , and $\sigma^\#(f(t_1, \dots, t_n)) = f(\sigma^\#(t_1), \dots, \sigma^\#(t_n))$.

Given two substitutions σ from X to Y and τ from Y to Z , their *composition* (denoted by $\tau \circ \sigma$) is the substitution from X to Z defined as $\tau \circ \sigma(x) = \tau^\#(\sigma(x))$. In the rest of the paper the extension of a substitution σ to terms will be denoted improperly by σ itself.

A substitution σ from X to Y is a *variable renaming* if $\sigma(x) \in Y$ for each $x \in X$ and σ is a bijection. If T and T' are two sets of terms we will write $T \cong T'$ if they are identical up to a variable renaming.

Given two substitutions σ and τ , σ is *more general* than τ if there exists a substitution ϕ such that $\phi \circ \sigma = \tau$. Two terms t and t' *unify* if there exists a substitution σ such that $\sigma(t) = \sigma(t')$. In this case σ is called a *unifier* of A and B . The set of unifiers of any two terms is either empty or it has a most general element (up to variable renaming) called the *most general unifier (mgu)*.

If G is a jungle, it should be quite obvious (looking, for example, at the jungle G of Example 1.4) how to extract a term of sort s from each node of G labeled by s . Indeed, by Definition 1.3, each node has either exactly one outgoing tentacle or none. In the former case it is the root of a subjungle which represents a term, while in the latter case it represents a variable.

Definition 1.9 (*From nodes to terms*). Let G be a jungle (over $\underline{\Sigma}$). The *variables of G* are a family of S -indexed sets $Var(G) = \{Var_s(G)\}_{s \in S}$ defined as

$$Var_s(G) = \{v \in V_G \mid outdegree_G(v) = 0 \text{ and } l_G(v) = s\}.$$

The function $term_G$ associates with each node of G labeled by s a term in $T_{\underline{\Sigma}_s}(Var(G))$. It is defined inductively as follows:

- $term_G(v) = v$ if $v \in Var_s(G)$,
- $term_G(v) = op(term_G(v_1), \dots, term_G(v_n))$ if there exists an $e \in E_G$ with $s_G(e) = v$, $t_G(e) = v_1 \dots v_n$, and $m_G(e) = op \in \Sigma_{w,s}$.

$TERM(G) = \bigcup_{s \in S} TERM_s(G)$ will denote the set of all terms associated with the nodes of jungle G , where $TERM_s(G) = \{term_G(v) \mid v \in V_G \text{ and } l_G(v) = s\}$. For each jungle G , the set $TERM(G)$ is clearly closed under the subterm relation, i.e., if $t \in TERM(G)$ and t' is a subterm of t , then also $t' \in TERM(G)$. If G and G' are two isomorphic jungles, then we have $TERM(G) \cong TERM(G')$.

Example 1.10 (*Terms associated with a jungle*). Consider the jungle G of Example 1.4. It contains just one variable v_3 of sort nat ; thus, $Var_{nat}(G) = \{v_3\}$ and $Var_{list}(G) = \emptyset$. The function $term_G$ is defined as $term_G(v_0) = cons(succ(v_3), cons(v_3, EMPTY))$; $term_G(v_1) = succ(v_3)$; $term_G(v_2) = cons(v_3, EMPTY)$; $term_G(v_4) = EMPTY$; and $term_G(v_3) = v_3$.

While the collection of terms associated with a jungle is uniquely determined, each set of terms (closed under the subterm relation) may have many representations, since identical subterms can either be collapsed into a single subjungle or not (cf. [30, 28, 20]). Nevertheless, a least representation as jungle always exists, and it is characterized by the fact that function $term$ is injective. The proof can be found in [20], or in [5] for the equivalent case of dags.

Definition 1.11 (*Representing a set of terms by a jungle*). Let T be a set of terms and let \bar{T} be the closure of T under the subterm relation, i.e.,

$$\bar{T} = \{t' \mid \text{there is a term } t \in T \text{ such that } t' \text{ is a subterm of } t\}.$$

Then a jungle G represents T iff $TERM(G) \cong \bar{T}$.

Note that we require just an equivalence up to variable renaming. This is because we consider variables just as placeholders, the names of which are not meaningful.

This fact implies that if a jungle G represents a set of terms T , then any other G' isomorphic to G represents T as well.

Proposition 1.12 (The fully collapsed jungle representing a set of terms). *Let T be a set of terms. Then the collection of jungles representing T has a final element $\mathcal{J}(T)$, called the fully collapsed jungle of T . That is, for every jungle G which represents T there exists a unique jungle morphism $G \rightarrow \mathcal{J}(T)$. The jungle $\mathcal{J}(T)$ is uniquely determined up to isomorphism, and it is such that the function $\text{term}_{\mathcal{J}(T)}: V_{\mathcal{J}(T)} \rightarrow T_{\Sigma}(Var(\mathcal{J}(T)))$ is injective.*

Just as jungles represent naturally collections of terms, so jungle morphisms correspond to term substitutions.

Definition 1.13 (The substitution induced by a morphism). Let G and G' be jungles, and $h: G \rightarrow G'$ be a jungle morphism. Then the node component of h (i.e., $h_V: V_G \rightarrow V_{G'}$) induces a substitution $\sigma_h: Var(G) \rightarrow T_{\Sigma}(Var(G'))$, defined as

$$\sigma_h(x) = \text{term}_{G'}(h_V(x)).$$

By Definition 1.13 it is easy to check that $\sigma_h \circ \text{term}_G = \text{term}_{G'} \circ h_V$. Moreover, the correspondence between jungle morphisms and term substitutions is preserved by composition, in the following sense: if $h: G \rightarrow G'$ and $k: G' \rightarrow G''$ are jungle morphisms then $\sigma_{k \circ h} = \sigma_k \circ \sigma_h: Var(G) \rightarrow T_{\Sigma}(Var(G''))$.

A morphism is uniquely determined by its induced substitution if the target jungle is fully collapsed. Furthermore, in that case the existence of a morphism can be checked by considering just the represented terms.

Proposition 1.14 (From substitution to morphisms). (1) *If G' is a fully collapsed jungle and $f, g: G \rightarrow G'$ are two morphisms such that $\sigma_f = \sigma_g$, then $f = g$.*

(2) *Given two jungles G and G' (with G' fully collapsed) and a substitution $\sigma: Var(G) \rightarrow T_{\Sigma}(Var(G'))$, there exists a morphism $h: G \rightarrow G'$ such that $\sigma_h = \sigma$ if and only if $\forall t \in TERM(G), \sigma(t) \in TERM(G')$.*

Proof. (1) By Definition 1.13 and the hypothesis, $\text{term}_{G'} \circ f_V = \sigma_f \circ \text{term}_G = \sigma_g \circ \text{term}_G = \text{term}_{G'} \circ g_V$, which implies $f_V = g_V$ because $\text{term}_{G'}$ is injective, by Proposition 1.12. Therefore $f = g$, by Fact 1.6.

(2) *If:* Define $h_V(v) = v'$, where v' is the unique node of G' such that $\text{term}_{G'}(v') = \sigma(\text{term}_G(v))$, by the injectivity of $\text{term}_{G'}$. On edges the definition of h is uniquely determined by the requirement that $s_{G'}(h_E(e)) = h_V(s_G(e))$, and it is easy to verify that the resulting h is indeed a graph morphism with $\sigma_h = \sigma$.

Only if: The proof is immediate, exploiting the fact that $\sigma_h \circ \text{term}_G = \text{term}_{G'} \circ h_V$. \square

1.3. Categorical constructions in the category of jungles

In the theory of graph grammars a central role is played by two basic operations on graphs: the *gluing* of two graphs, which merges the graphs together identifying some selected edges and nodes; and the deletion of a subgraph from a graph. These operations are cleanly formulated, in the algebraic approach to graph grammars, as simple categorical constructions, namely, the *pushout* and the *pushout complement*.

In this section we establish necessary and sufficient conditions for the existence of pushouts in the category of jungles (generalizing a similar result in [5]), and we show that the pushout complement always exists under hypotheses which will always be satisfied in the cases we are interested in. For the pushouts it is interesting to note that although the category $\mathbf{HGraph}_{\underline{\Sigma}}$ has all pushouts, they do not always exist in its full subcategory $\mathbf{Jungle}_{\underline{\Sigma}}$; indeed, it is known in the “folklore” that the pushout in such categories (as for dags) corresponds to the existence of most general unifiers (see, for example, [28]). We make this statement precise.

Definition 1.15 (*Pushout and pushout complement*). Given a category C and two arrows $b: K \rightarrow B$, $d: K \rightarrow D$ of C , a triple $\langle H, h: B \rightarrow H, c: D \rightarrow H \rangle$ is called a *pushout* of $\langle b, d \rangle$ if

- *commutativity property*: $h \circ b = c \circ d$;
- *universal property*: for all objects H' and arrows $h': B \rightarrow H'$ and $c': D \rightarrow H'$, with $h' \circ b = c' \circ d$, there exists a unique arrow $f: H \rightarrow H'$ such that $f \circ h = h'$ and $f \circ c = c'$, as in Fig. 2.

In this situation, H is called a *pushout object* of $\langle b, d \rangle$. Moreover, given arrows $b: K \rightarrow B$ and $h: B \rightarrow H$, a *pushout complement* of $\langle b, h \rangle$ is a triple $\langle D, d: K \rightarrow D, c: D \rightarrow H \rangle$ such that $\langle H, h, c \rangle$ is a pushout of $\langle b, d \rangle$. In this case D is called a *pushout complement object* of $\langle b, h \rangle$.

Example 1.16 (*Pushouts in Set and in $\mathbf{HGraph}_{\underline{\Sigma}}$*). The paradigmatic example of category is \mathbf{Set} , i.e., the category having sets as objects and total functions as arrows. It is easy to show that the pushout of two arrows in \mathbf{Set} always exists, and that it is characterized (up to isomorphism) as follows. If $b: K \rightarrow B$ and $d: K \rightarrow D$ are two functions, then the pushout of $\langle b, d \rangle$ is the set $H \triangleq (B + D) / \approx$ (i.e., the quotient set of the disjoint union of B and D , modulo the equivalence relation “ \approx ”, which is the

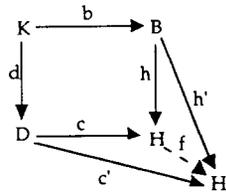


Fig. 2.

least equivalence relation such that for all $k \in K$, $b(k) \approx d(k)$, together with the two functions $h: B \rightarrow H$ and $c: D \rightarrow H$ that map each element to its equivalence class.

In category $\mathbf{HGraph}_{\underline{\Sigma}}$ the pushout of two arrows always exists as well: it can be computed componentwise (as a pushout in \mathbf{Set}) for the nodes and for the edges, and the source, target, and labeling mappings are uniquely determined. More precisely, if $X = (V_X, E_X, s_X, t_X, m_X, l_X)$ for $X \in \{K, B, D, H\}$ are objects of $\mathbf{HGraph}_{\underline{\Sigma}}$, and $b = (b_V, b_E): K \rightarrow B$ and $d = (d_V, d_E): K \rightarrow D$ are graph morphisms, then the pushout object H of $\langle b, d \rangle$ is as follows.

- V_H is the pushout in \mathbf{Set} of $b_V: V_K \rightarrow V_B$ and $d_V: V_K \rightarrow V_D$,
- E_H is the pushout in \mathbf{Set} of $b_E: E_K \rightarrow E_B$ and $d_E: E_K \rightarrow E_D$,
- $l_H([v]) = \begin{cases} l_B(v) & \text{if } v \in V_B, \\ l_D(v) & \text{otherwise,} \end{cases}$
- $m_H([e]) = \begin{cases} m_B(e) & \text{if } e \in E_B, \\ m_D(e) & \text{otherwise,} \end{cases}$
- $s_H([e]) = \langle [v_1], \dots, [v_n] \rangle$, where $s_B(e) = \langle v_1, \dots, v_n \rangle$ if $e \in E_B$ and $s_D(e) = \langle v_1, \dots, v_n \rangle$ if $e \in E_D$,
- $t_H([e]) = \langle [v_1], \dots, [v_n] \rangle$, where $t_B(e) = \langle v_1, \dots, v_n \rangle$ if $e \in E_B$ and $t_D(e) = \langle v_1, \dots, v_n \rangle$ if $e \in E_D$.

It can be shown easily that this definition is correct, i.e., it does not depend on the choice of the representative of an equivalence class.

In any category, if the pushout object of two arrows exists then it is unique up to a unique isomorphism because of the universal property. On the contrary, since the pushout complement object is not characterized directly by a universal property, for a pair of arrows there can exist many pushout complement objects which are not isomorphic, if any. For example, consider Fig. 3 in the category of sets and functions. For the left diagram there exist no set and arrows which can close the square making it a pushout, while in the right diagram there exist two pushout complements D and D' which are not isomorphic.

In the category of jungles, the pushout of two arrows exists iff the associated substitutions have an unifier. Before stating this result, we present a lemma which warrants the existence of a pushout in terms of a weaker condition, i.e., the existence of

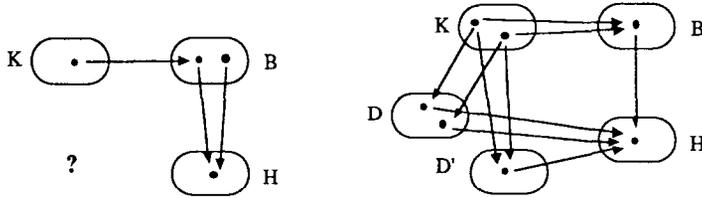


Fig. 3.

at least one commutative diagram. The lemma is a consequence of a more general result (cited in [24]) stating that \mathbf{Jungle}_{Σ} has all colimits over diagrams for which a cocone exists. Nevertheless, our explicit proof is an example of a fruitful technique, which consists of temporarily “forgetting” the additional requirements for jungles in order to exploit the good properties of the super-category \mathbf{HGraph}_{Σ} . Moreover, the proof hints how to perform unification directly on jungles.

Lemma 1.17 (Existence of pushouts in \mathbf{Jungle}_{Σ}). *Let $r: K \rightarrow R$ and $d: K \rightarrow D$ be two morphisms in \mathbf{Jungle}_{Σ} . Then the pushout of $\langle r, d \rangle$ exists iff there are two jungle morphisms $l: R \rightarrow X$, $k: D \rightarrow X$ such that $l \circ r = k \circ d$.*

Proof. The *only if* part is obvious by the commutativity of pushouts.

If: Since jungles are hypergraphs, let H (with injections $f: R \rightarrow H$, $g: D \rightarrow H$) be the pushout object of $\langle r, d \rangle$ in category \mathbf{HGraph}_{Σ} , which always exists (as discussed in Example 1.16).

By the universal property of H , there exists a unique arrow $c: H \rightarrow X$. By exploiting the existence of c , we construct a jungle J (which is a suitable quotient of H) and a surjective morphism $h: H \rightarrow J$ such that c factorizes through h , i.e., $c = x \circ h$ for some x (as in Fig. 4). Since the construction exploits just the existence of c , the fact that J (with injections $h \circ f$ and $h \circ g$) is the pushout in \mathbf{Jungle}_{Σ} of $\langle r, d \rangle$ easily follows: indeed, the square obviously commutes, and if $l': R \rightarrow X'$ and $k': D \rightarrow X'$ form another commutative square based on $\langle r, d \rangle$, then $x': J \rightarrow X'$ is obtained by the factorization of the unique $c': H \rightarrow X'$ through h ; x' is a jungle morphism (like every graph morphism between two jungles), and is unique because h is surjective.

It remains to construct J , $h: H \rightarrow J$ and $x: J \rightarrow X$. They are defined as the last element of a finite sequence of triple $\langle \langle id_H, H, c \rangle \triangleq \langle h_0, J_0, c_0 \rangle, \dots, \langle h_n, J_n, c_n \rangle \triangleq \langle h, J, x \rangle \rangle$, where for each i , $h_i: H \rightarrow J_i$ is surjective and $c_i: J_i \rightarrow X$ (this is clearly true for $n=0$). The triple $\langle h_{i+1}, J_{i+1}, c_{i+1} \rangle$ is obtained from $\langle h_i, J_i = (V_i, E_i, s_i, t_i, m_i, l_i), c_i \rangle$ as follows.

- If the set of nodes of J_i with outdegree greater than or equal to two, $OUT \geq 2(J_i)$, is empty, then stop: J_i is the searched jungle and h_i and c_i satisfy the requirements for h and x . Indeed, J_i is acyclic (otherwise its image through c_i in X would be cyclic too), and its labeling is consistent with the signature for a similar reason.

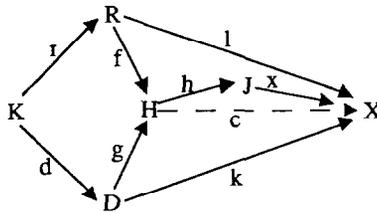


Fig. 4.

- If $OUT \geq 2(J_i) \neq \emptyset$, then by the acyclicity of J_i there exists at least one node $\underline{v} \in V_i$ in $OUT \geq 2(J_i)$ such that there is no (nonempty) path from any other node in $OUT \geq 2(J_i)$ to \underline{v} . Then J_{i+1} is obtained from J_i by identifying all the edges having \underline{v} as source, and by adjusting the other components correspondingly. Formally, J_{i+1} is defined as follows: $E_{i+1} = E_i \setminus \approx_E$, where $e \approx_E e'$ if $s_i(e) = s_i(e') = \underline{v}$; $V_{i+1} = V_i \setminus \approx_V$, where $v_k \approx_V v'_k$ if there exist e, e' such that $e \approx_E e'$, $t_i(e) = v_1 \dots v_n$ and $t_i(e') = v'_1 \dots v'_n$. The source and target functions are defined as $s_{i+1}([e]) = [s_i(e)]$ and $t_{i+1}([e]) = [v_1] \dots [v_n]$, if $t_i(e) = v_1 \dots v_n$ (and are clearly well defined). For the labeling, define $m_{i+1}([e]) = m_i(e)$ and $l_{i+1}([v]) = l_i(v)$. Indeed, m_i and l_i are consistent w.r.t. the equivalence classes, thanks to the existence of $c_i: J_i \rightarrow X$. In fact, if $e \approx_E e'$ we must have $c_{iE}(e) = c_{iE}(e')$ (else X would not be a jungle, since both $c_{iE}(e)$ and $c_{iE}(e')$ have the same source node $c_{iV}(v)$); thus, $m_i(e) = m_X(c_{iE}(e)) = m_X(c_{iE}(e')) = m_i(e')$. Similarly, if $v' \approx_V v''$ then $l_i(v') = l_i(v'')$. The last argument also shows that c_i is consistent with the equivalence classes; thus, we define $c_{i+1}: J_{i+1} \rightarrow X$ as $c_{i+1}([x]) = [c_i(x)]$ on both nodes and edges. Finally, by construction, there is a surjective morphism $j_i: J_i \rightarrow J_{i+1}$ mapping each item to its equivalence class: then $h_{i+1}: H \rightarrow J_{i+1}$ is defined as $j_i \circ h_i$.

The sequence of triples is clearly finite because of the finiteness and acyclicity of H . Moreover, the morphism h and the jungle J do not depend on c , in the sense that any other graph morphism $c': H \rightarrow X'$ would lead to the same h and J , as it is manifest from the construction. \square

As an example of the construction just described, consider Fig. 5, where K contains a single node, the jungles R and D are isomorphic and represent the term $f^n(x)$ for a given natural number n , and morphisms $r: K \rightarrow R$ and $d: K \rightarrow D$ are as depicted. It is easy to see that the hypothesis of Lemma 1.17 is satisfied, that is there exist jungle morphisms $l: R \rightarrow X$, $k: D \rightarrow X$ such that $l \circ r = k \circ d$ (take $X = \mathcal{J}(T)$ for any set of term T which contains $f^n(x)$, and define l and k with care).

In Fig. 5 we depicted the pushout object of $\langle r, d \rangle$ in the category of hypergraphs, H , which clearly is not a jungle. Applying the construction defined in the proof of Lemma 1.17 to hypergraph H , at each step two nodes will be identified (moving from the root towards the leaves) and in n steps the pushout object jungle will be produced, which is obviously isomorphic to R and D . In particular, as the result of the i th step ($0 \leq i \leq n$), the hypergraph J_i mentioned in the proof has the shape shown in Fig. 6, which substantiates the intuition that the construction performs a “folding” of graph structures.

The next result generalizes a similar result in [5], where graph K was assumed to be discrete.

Proposition 1.18 (Pushout in $\mathbf{Jungle}_{\underline{\Sigma}}$ as most general unifier). (1) *Let $r: K \rightarrow R$ and $d: K \rightarrow D$ be two jungle morphisms. Let $\sigma_r: \text{Var}(K) \rightarrow T_{\underline{\Sigma}}(\text{Var}(R))$ and $\sigma_d: \text{Var}(K) \rightarrow T_{\underline{\Sigma}}(\text{Var}(D))$ be the associated substitutions (see Definition 1.13). Then the pushout of $\langle r, d \rangle$ exists in $\mathbf{Jungle}_{\underline{\Sigma}}$ if and only if there exist two substitutions $\theta: \text{Var}(R) \rightarrow T_{\underline{\Sigma}}(Y)$ and $\theta': \text{Var}(D) \rightarrow T_{\underline{\Sigma}}(Y)$ which “unify” $\langle \sigma_r, \sigma_d \rangle$, in the sense that $\theta \circ \sigma_r = \theta' \circ \sigma_d$.*

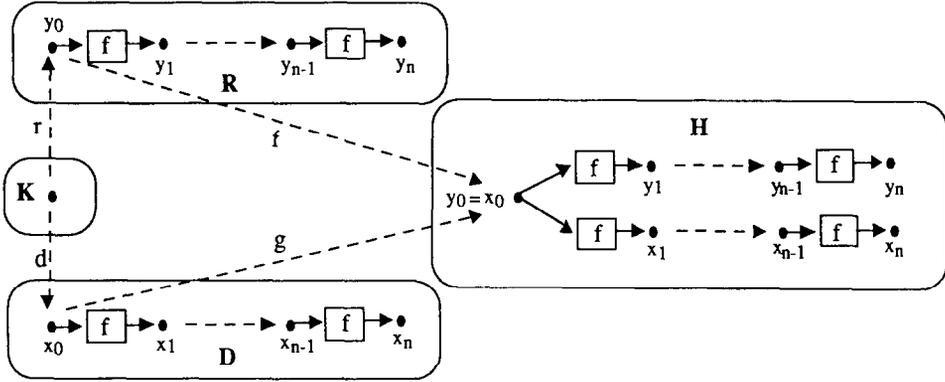


Fig. 5.

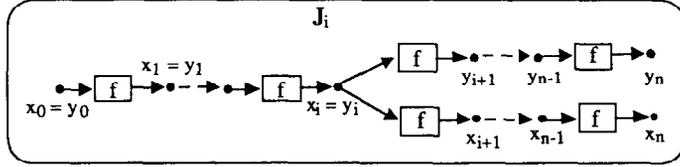


Fig. 6.

$$\begin{array}{ccc}
 K & \xrightarrow{r} & R \\
 d \downarrow & & \downarrow g \\
 D & \xrightarrow{f} & H
 \end{array}$$

Fig. 7.

(2) If $\langle g, f \rangle$ is a pushout of $\langle r, d \rangle$ in \mathbf{Jungle}_{Σ} , as in Fig. 7, then $\langle \sigma_g, \sigma_f \rangle$ is a most general unifier of $\langle \sigma_r, \sigma_d \rangle$, i.e., for each pair of substitutions $\langle \theta, \theta' \rangle$ such that $\theta \circ \sigma_r = \theta' \circ \sigma_d$, there exists a σ such that $\sigma \circ \sigma_g = \theta$ and $\sigma \circ \sigma_f = \theta'$. Moreover, jungle H is such that $TERM(H) = \sigma_f(TERM(D)) \cup \sigma_g(TERM(R))$.

Proof. (1) *Only if:* Let $\langle g: R \rightarrow H, f: D \rightarrow H \rangle$ be a pushout of $\langle r, d \rangle$. Then by commutativity of pushouts and by Definition 1.13 we have $\sigma_g \circ \sigma_r = \sigma_{g \circ r} = \sigma_{f \circ d} = \sigma_f \circ \sigma_d$.

If: Suppose that $\theta: Var(R) \rightarrow T_{\Sigma}(Y)$ and $\theta': Var(D) \rightarrow T_{\Sigma}(Y)$ are such that $\theta \circ \sigma_r = \theta' \circ \sigma_d$. Then let $X = \mathcal{J}(\theta(TERM(R)) \cup \theta'(TERM(D)))$, and $l: R \rightarrow X$, $k: D \rightarrow X$ be the unique morphisms such that $\sigma_l = \theta$ and $\sigma_k = \theta'$, by Proposition 1.14. Then by the same proposition we have $l \circ r = k \circ d$, and by Lemma 1.17 the pushout of $\langle r, d \rangle$ in \mathbf{Jungle}_{Σ} exists.

(2) Let $\langle \theta : \text{Var}(R) \rightarrow T_{\Sigma}(Y), \theta' : \text{Var}(D) \rightarrow T_{\Sigma}(Y) \rangle$ be a pair of substitutions such that $\theta \circ \sigma_r = \theta' \circ \sigma_d$, and let $X, l : R \rightarrow X$, and $k : D \rightarrow X$ be as above, with $\sigma_l = \theta$ and $\sigma_k = \theta'$. Since $l \circ r = k \circ d$, there exists a unique morphism $h : H \rightarrow X$, and by Definition 1.13 $\sigma_h \circ \sigma_g = \theta$ and $\sigma_h \circ \sigma_f = \theta'$.

For the last point, it is easy to check that $\text{TERM}(H) \supseteq \sigma_f(\text{TERM}(D)) \cup \sigma_g(\text{TERM}(R))$ using Definition 1.13. The other inclusion holds because by the universal property of H there must exist a morphism from H to $X' = \mathcal{J}(\sigma_f(\text{TERM}(D)) \cup \sigma_g(\text{TERM}(R)))$, because there are morphisms $l' : R \rightarrow X'$ and $k' : D \rightarrow X'$ such that the resulting square commutes. \square

It is worth noting that the relationship between most general unifiers and universal constructions in a category has been stressed in many places. For example, mgu's are characterized in [31] ([17]) in terms of coequalizers (equalizers, due to the dual approach), and also as pullbacks in [1]; in these papers terms and substitutions are represented as arrows of a category. On the contrary, our characterization is much closer to the one in [28, 5], where terms are objects and mgu's are pushouts.

The characterization of sufficient conditions for the existence and uniqueness of the pushout complement of two arrows is a central topic in the theory of graph grammars, because it allows one to check the applicability of a rewrite rule to a graph. However, since in this paper we are interested just in the application of the so-called hyperedge replacement rewrite rules (as defined in Definition 2.1), we consider just the pushout complements which arise from the application of a rewrite rule of that type, showing that in this case, although the pushout complement is not unique, a “minimal” pushout complement can always be characterized.

Proposition 1.19 (Existence of certain PO-complements in **Jungle** $_{\Sigma}$). *Let $l : K \rightarrow L$ and $g : L \rightarrow G$ be two jungle morphisms such that*

- *jungle L consists of exactly one edge, say with label f , connected to $n+1$ distinct nodes, where n is the rank of f ;*
- *jungle K is discrete and includes exactly $n+1$ nodes; and*
- *the morphism $l : K \rightarrow L$ is a bijection on nodes.*

Then there exist exactly two nonisomorphic pushout complements of $\langle l, g \rangle$, say $\langle D, k : K \rightarrow D, d : D \rightarrow G \rangle$ and $\langle D', k' : K \rightarrow D', d' : D' \rightarrow G \rangle$, where both d and d' are injective and are one-to-one on nodes. Moreover, there is an injective morphism from one pushout complement object to the other, say $i : D \rightarrow D'$.

Proof (sketch). Figure 8 summarizes a paradigmatic situation. The top row shows morphism $l : K \rightarrow L$, and the image of L in G through g is explicitly depicted. Clearly, we can have $g_v(v_i) = g_v(v_j)$ with $i \neq j$ (but $v_i \neq v_j$, by definition). Dashed thin arrows in jungles G, D, D' represent possible further connections between the relevant part and the rest of those jungles. Jungle D' is isomorphic to G , while D is obtained by removing from G the unique edge in the image of g ; thus, $i : D \rightarrow D'$ is the obvious inclusion. D and

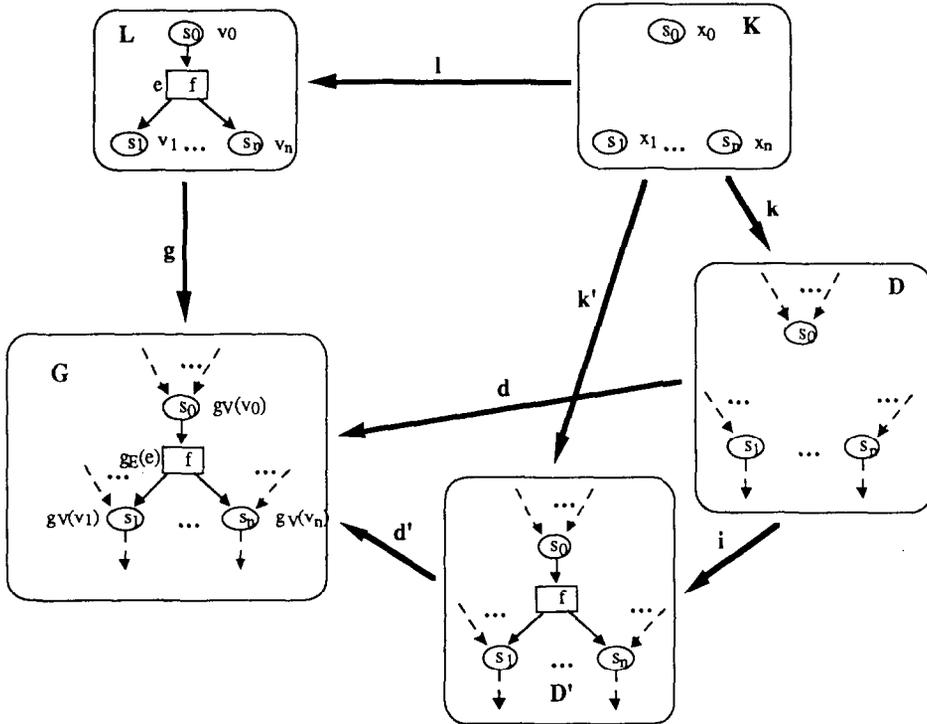


Fig. 8.

D' are determined up to isomorphism and both make the resulting square a pushout. Since D has one edge less than D' , we qualify it as the “smallest” pushout complement object. \square

2. Hyperedge replacement jungle rewriting

In this section we introduce the basic concepts of the algebraic theory of graph rewriting [16, 11] for the specific case of the category of jungles. Besides some basic notions (including rewrite rules, rewriting, grammars and so on) which could be defined for any category in terms of suitable diagrams and constructions [12], we also recall some definitions which only make sense in the case of jungles, like the track function [21, 22] and the substitution [5] associated with rewriting steps. A relevant point of this presentation is the commitment to hyperedge replacement jungle rewrite rule (similar, at least syntactically, to the hyperedge replacement hypergraph rules presented in [3, 18]) which is further discussed in Section 2.2.

2.1. Rewriting in the category of jungles

A jungle rewrite rule (analogously to term rewrite rules of string productions) describes how to replace the occurrence of a subgraph L in a jungle G with another jungle R . While in the case of terms or strings the embedding of R inside G is uniquely determined, this is not true in the more general case of graphical structures like jungles. Thus, a third graph K is needed in order to give the connection points of R in G . In this paper we will restrict our attention to hyperedge replacement rewrite rules, i.e., rules where the jungle on the left-hand side contains exactly one edge. In this section we introduce both the terminology of graph-rewriting systems and that of graph grammars in parallel: indeed, the first one is more suitable for describing TRSs, while for logic programming the second one is more natural.

Definition 2.1 (*Hyperedge replacement jungle rewrite rules*).

A jungle (rewrite) rule p is a pair of jungle morphisms

$$L \xleftarrow{l} K \xrightarrow{r} R,$$

where l is injective. L , K , and R are called the left-hand side (lhs), the interface, and the right-hand side (rhs) of p , respectively.

A *hyperedge replacement* (shortly HR) jungle rewrite rule is such that L , K and l satisfy the hypotheses of Proposition 1.19, i.e., L has just one edge (say with label f) connected to $n+1$ distinct nodes, where n is the rank of f , K is discrete and has $n+1$ nodes, and l is one-to-one on nodes. In a hyperedge replacement rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ the jungles L , K and the morphism l are uniquely determined (up to isomorphisms) by the label of the unique edge of L . As a consequence, we will often represent p in a more compact way as

$$p: f \rightsquigarrow (R, v_0, \dots, v_n),$$

where f is the label of the edge of L , and v_0, \dots, v_n are distinguished nodes of R (possibly with $v_i = v_j$ for some $i \neq j$) which are the images through r of the $n+1$ nodes of K . We assume that nodes v_0, \dots, v_n correspond to the source and to the n target nodes of the unique edge of L , in this order. Formally, if $V_K = \{x_0, \dots, x_n\}$ and $E_L = \{e\}$, we have $s_L(e) = l_V(x_0)$, $t_L(e) = l_V(x_1) \dots l_V(x_n)$ and $r_V(x_i) = v_i$ for $0 \leq i \leq n$.

Examples of hyperedge replacement jungle rules will be given in Sections 3 and 4. The application of a rewrite rule p to a jungle G is modeled by a double pushout construction, following the classical algebraic approach to graph grammars [11].

Definition 2.2 (*Direct rewriting*). Given a jungle G , a jungle rewrite rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$, and an occurrence (i.e., a jungle morphism) $g: L \rightarrow G$, a *direct rewriting* (or *direct derivation*) from G to H based on g exists if and only if the diagram in Fig. 9 can be constructed, where both squares are required to be pushouts in **Jungle** _{$\underline{\Sigma}$} . In this case, D is called the *context jungle*, and we write $G \Rightarrow_p H$.

$$\begin{array}{ccccc}
 L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
 \downarrow g & & \downarrow k & & \downarrow h \\
 G & \xleftarrow{d} & D & \xrightarrow{b} & H
 \end{array}$$

Fig. 9. *Double-pushout diagram.*

The double pushout construction can be interpreted as follows. In order to apply the rule p to G , we first need to find an occurrence of its lhs L in G , i.e., a jungle morphism $g: L \rightarrow G$. Next, to model the deletion of that occurrence from G , we construct the pushout complement of g and l , producing the context jungle D , where the rhs R has to be embedded. This embedding is expressed by the second pushout.

It must be noted that this construction can fail, since the required pushout and pushout complement do not always exist in the category of jungles. Nevertheless, in the case of a hyperedge replacement direct rewriting, since arrow $l: K \rightarrow L$ satisfies the hypotheses of Proposition 1.19, there are always two nonisomorphic pushout complement objects, one of which is included in the other. As a consequence, the application of a hyperedge replacement rewrite rule requires just the existence of the right-hand side pushout.

Looking at the proof of Proposition 1.19, it should be clear that just the smallest pushout complement deletes the occurrence of L from G , and it is, therefore, the one we will choose. Actually, the main results of Sections 3 and 4 are based on the following assumption.

Assumption 2.3 (*Always take the smallest pushout complement object*). Whenever we construct a direct rewriting based on a hyperedge replacement rule, we always assume to take as pushout complement object of $\langle l, g \rangle$ the smallest one among the two which exist by Proposition 1.19.

In the case of a hyperedge replacement direct rewriting $G \Rightarrow_p H$ there is a *track function* [21, 22] from the nodes of G to the nodes of H , which induces a substitution [5].

Definition 2.4 (*Track function and substitution of a direct rewriting*). Let p be a HR jungle rule and $G \Rightarrow_p H$ be a direct rewriting. By Proposition 1.19, morphism $d: D \rightarrow G$ is a bijection on nodes. Then the *track function associated with $G \Rightarrow_p H$* is defined as $tr = b_V \circ d_V^{-1}: G_V \rightarrow H_V$. The *substitution associated with $G \Rightarrow_p H$* is the one induced by the track function (i.e., $\sigma_{tr}: Var(G) \rightarrow T_{\Sigma}(Var(H))$) in the sense of Definition 1.13 (see Fig. 10).

Definition 2.5 (*Jungle-rewriting systems and jungle grammars*). A *jungle-rewriting system* (over Σ) is a finite set \mathcal{R} of jungle rewrite rules in category \mathbf{Jungle}_{Σ} . A *jungle*

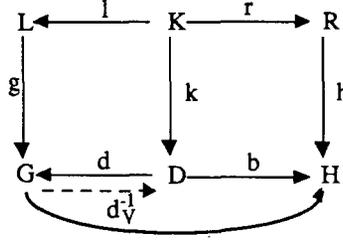


Fig. 10. Track: $tr = b_v \circ d_v^{-1} : G_v \rightarrow H_v$; substitution: $\sigma_{tr} : Var(G) \rightarrow T_{\underline{\Sigma}}(Var(H))$.

grammar is a four-tuple $\mathcal{G} = (\mathcal{R}, G_0, \underline{\Sigma}, \underline{\Phi})$, where \mathcal{R} is a rewriting system over $\underline{\Sigma}$, G_0 is an object of $\mathbf{Jungle}_{\underline{\Sigma}}$ (called the *initial jungle*), and $\underline{\Phi}$ (called the *terminal colors*) is a *subsignature* of $\underline{\Sigma}$, i.e., if $\underline{\Sigma} = (S, \{\Sigma_{w,s}\})$ and $\underline{\Phi} = (F, \{\Phi_{v,f}\})$, then $F \subseteq S$ and $\Phi_{v,f} \subseteq \Sigma_{v,f}$ for all $\langle v, f \rangle \in F^* \times F$.

A *hyperedge replacement jungle-rewriting system* \mathcal{R} is a jungle-rewriting system where all the rewrite rules are hyperedge replacement rules. A *hyperedge replacement jungle grammar* $\mathcal{G} = (\mathcal{R}, G_0, \underline{\Sigma}, \underline{\Phi})$ is a jungle grammar such that \mathcal{R} is a HR jungle-rewriting system and, moreover, for each rewrite rule $p = (f \rightsquigarrow (R, v_0, \dots, v_n))$ in \mathcal{R} the operator f is not in $\underline{\Phi}$.

Definition 2.6 (*Rewriting and derivations*). Given two jungles G and H and a rewriting system \mathcal{R} , a *rewriting* from G to H over \mathcal{R} , denoted by $G \Rightarrow_{\mathcal{R}}^* H$, is a finite sequence of direct rewriting steps of the form $G \Rightarrow_{p_1} G_1 \Rightarrow_{p_2} \dots \Rightarrow_{p_n} G_n = H$, where p_1, \dots, p_n are in \mathcal{R} .

Given a jungle grammar $\mathcal{G} = (\mathcal{R}, G_0, \underline{\Sigma}, \underline{\Phi})$, a *derivation* from G to H over \mathcal{G} ($G \Rightarrow_{\mathcal{G}}^* H$) is a rewriting from G to H over \mathcal{R} such that $G \equiv G_0$ and all the labels of H belong to $\underline{\Phi}$.

The track functions and their induced substitutions introduced in Definition 2.4 can be extended to an entire rewriting or derivation.

Definition 2.7 (*Track function and substitution associated with derivations*). If $\rho = (D_0 \Rightarrow_{p_1} D_1 \Rightarrow_{p_2} \dots \Rightarrow_{p_n} D_n)$ is a rewriting and tr_i is the track function associated with $D_{i-1} \Rightarrow_{p_i} D_i$ for each $1 \leq i \leq n$, then the *track function associated with ρ* is their composition $tr = tr_n \circ \dots \circ tr_1 : D_{0V} \rightarrow D_{nV}$. Similarly, the *substitution associated with ρ* is $\sigma_{tr} : Var(D_0) \rightarrow T_{\underline{\Sigma}}(Var(D_n))$, or equivalently $\sigma_{tr} = \sigma_{tr_n} \circ \dots \circ \sigma_{tr_1}$.

Finally, as for strings, each jungle grammar implicitly defines a (possibly infinite) set of jungles (its language) which includes all the terminal jungles derivable from the initial jungle.

Definition 2.8 (*The language generated by a jungle grammar*). The *language $\mathcal{L}(\mathcal{G})$* generated by a jungle grammar $\mathcal{G} = (\mathcal{R}, G_0, \underline{\Sigma}, \underline{\Phi})$ is the set of jungles labeled by terminal colors derivable from G_0 , i.e., $\mathcal{L}(\mathcal{G}) = \{H \mid G_0 \Rightarrow_{\mathcal{G}}^* H\}$.

2.2. Hyperedge replacement rules and their applicability

Hyperedge replacement hypergraph rewriting is the central topic of the thesis of Habel [18] (see also [19]), and has been introduced independently by Bauderon and Courcelle [3] in the framework of their axiomatization of graphs and graph rewriting.

Looking at jungles as hypergraphs, our definition of hyperedge replacement rules essentially coincides (up to some minor details) with that in [3], and it is slightly more general than the one by Habel. In both [3] and [18], the hypergraphs are equipped with finite sequences of distinguished nodes (one sequence, called the *sources*, in [3], and two sequences, called *begin* and *end*, in [18]). Moreover, in a rewrite rule like $f \rightsquigarrow H$ (where f is a ranked symbol and H is a graph), the length(s) of the sequence(s) of distinguished nodes of H must be consistent with the rank of f . This requirement coincides with ours (Definition 2.1), where in the compact representation of a rule as $f \rightsquigarrow (R, v_0, \dots, v_n)$ we have to indicate explicitly $n + 1$ nodes of jungle R . In [18] it is also required that nodes appearing in the *begin* and *end* sequences of a graph be pairwise distinct, which is necessary neither in [3] nor in our definition.

In spite of the syntactical similarities, our HR jungle rules greatly differ with respect to the ones just mentioned, which will be called *HR hypergraph rules* in the following, both for the applicability conditions and for the effect they can have on a graph. As a consequence, they enjoy more expressive power, as it will be manifest by the treatment of term-rewriting systems and logic programming in Sections 3 and 4.

Both [18] and [3] define the basic notion of hyperedge replacement (called *graph substitution* in [3]) in a set-theoretic way, and then they prove that it can be formulated equivalently as a double pushout in the category of hypergraphs. From our side, we defined it directly in terms of a double pushout, but in the category of jungles (a set-theoretic definition is clearly possible, but it would not be manageable). The consequences of the change of the underlying category are fundamental, and are described in the following paragraphs.

Applicability conditions: A HR hypergraph rule $p: f \rightsquigarrow H$ can always be applied to a graph G provided that there is at least one edge labeled by f . In terms of double pushouts, this means that both the lhs pushout complement and the rhs pushout of Fig. 9 always exist. Actually, for the pushout complement a result stronger than Proposition 1.19 holds in the category of hypergraphs, since under the same hypotheses the pushout complement is unique; moreover, the pushout of any two arrows exists. On the contrary, constructing the double pushout in the category of jungles as we do, the existence of one edge of G labeled by f only warrants the existence of the pushout complement, but the existence of the rhs pushout has to be checked independently.

Effects produced by rule application: The modifications caused to a graph G by the application of a HR hypergraph rule p are “strictly local” in the case of [18] (i.e., two distinct nodes or edges of G are still distinct after the application of p) and “local” in [3] (two distinct nodes of G can be identified by the application of p , but only if they are directly connected to the edge of G which has been replaced). On the contrary,

because of the properties of pushouts in the category of jungles, the effect of the application of a HR jungle rule may not be local, in the sense that it can cause the identification of two edges or nodes at any distance from the replaced edge. This could be checked by looking at the example shown after Lemma 1.17, where the merging of two nodes causes the folding of an arbitrary deep graphical structure.

Expressive power. In the case of hypergraph rewriting it is evident that unrestricted rules have more expressive power than hyperedge replacement ones. For example, term-rewriting systems are modeled in [20–22] using hypergraph-rewriting rules which can have any number of hyperedges on their left-hand sides, and it does not seem possible to do it with HR hypergraph rewriting. Moreover, a HR hypergraph rule can be applied to any occurrence of its lhs in a graph, while the application of an unrestricted rule may fail if the pushout complement does not exist.

In the case of jungles, hyperedge replacement rewriting is powerful enough to model not only term-rewriting systems, but also logic programming (as shown in the rest of this paper). On the other hand, the application of a HR jungle rule can fail if the rhs pushout does not exist. It is not clear how much expressive power is lost with the restriction to HR jungle rules: the properties of unrestricted jungle rewriting have not been explored yet, to our knowledge.

It is worth noting that the restriction to hyperedge replacement rules partially justifies, a posteriori, our choice of jungles (instead of dags) for the representation of terms. Indeed, although all the formal results presented in the paper also hold for dags, the dag equivalent to the lhs jungle of a HR jungle rule contains in general more than one edge; thus the specification “hyperedge replacement” would be less justified in that case.

3. Term-rewriting systems

Since jungles naturally represent terms, it is not surprising that jungle-rewriting systems can be used to model term-rewriting systems, along the guidelines of the term graph rewriting tradition. Indeed, jungles have been originally defined with this aim in mind in [29, 20–22], thus providing a new contribution to the area of term graph rewriting, where terms were usually represented as dags ([30, 23, 28, 2] among others).

Here we propose a translation of term rewrite rules into hyperedge replacement jungle rules. Our representation substantially differs from all the others proposed in the literature, and it is more satisfactory in the sense that it allows one to manage non-left-linear rules uniformly, without resorting to additional mechanisms. In Section 3.1, after the basic definitions about term-rewriting systems, we define the jungle representation $\mathcal{J}(t \rightarrow t')$ of a term rewrite rule $t \rightarrow t'$.

Next, in Section 3.2, we show that the representation is sound, i.e., whenever $\mathcal{J}(t \rightarrow t')$ can be applied to a jungle G producing G' , then every term in $TERM(G')$ can be obtained by a finite number of applications of rule $t \rightarrow t'$ from a term in $TERM(G)$. After a short discussion about the problems which arise from the non-left-linearity of

rules, we also prove a result of completeness w.r.t. applicability of our representation, i.e., if term s can be rewritten using rule R , then there is a direct rewriting from $\mathcal{J}(s)$ using $\mathcal{J}(t \rightarrow t')$. The main point is that this result holds for non-left-linear rules as well. The comparison between our representation and some related works is discussed in more detail in Section 5.

Although the main topic of this section concerns the representation of TRSs by jungle rewriting, we also show in Section 3.3 how to go in the other direction, i.e., how to extract a term-rewriting system $\mathcal{T}(\mathcal{R})$ from a given hyperedge replacement jungle-rewriting system \mathcal{R} . The relationship between \mathcal{R} and $\mathcal{T}(\mathcal{R})$ will be briefly explored.

3.1. Jungle representation of term-rewriting systems

We first introduce our working definition of term-rewriting systems.

Definition 3.1 (*Term-rewriting systems*). A (term) rewrite rule (over a given signature $\underline{\Sigma} = (S, \Sigma)$) is a pair $t \rightarrow t'$ of terms such that $\text{sort}(t) = \text{sort}(t')$, t is not a variable, and all the variables in t' occur also in t . A rule $t \rightarrow t'$ is *left-linear* if all the variables occurring in t are distinct. A (term-) rewriting system (TRS) is a finite set of rewrite rules $\mathcal{T} = \{t_i \rightarrow t'_i\}_{i \leq n}$.

Given a rewriting system \mathcal{T} , a ground term s rewrites to a term s' (denoted by $s \rightarrow_{\mathcal{T}} s'$) iff there exists a rewrite rule $t \rightarrow t'$ in \mathcal{T} and a substitution σ such that s has a subterm $s'' = \sigma(t)$, and s' is obtained from s by replacing s'' by $\sigma(t')$. Clearly, in this case s' is ground, too. The reflexive and transitive closure of $\rightarrow_{\mathcal{T}}$ will be denoted by $\rightarrow_{\mathcal{T}}^*$.

For technical reasons we only consider the rewriting of *ground* terms, but this is not a real limitation. In fact, as far as the rewriting process involves just pattern matching and not full unification (as it is usual in the literature), the variables appearing in the term to be rewritten are never instantiated by the application of a rewrite rule and, thus, they can safely be considered as new constants.

Definition 3.2 (*Representing term rules by hyperedge replacement jungle rules*). Given a rewrite rule $t \rightarrow t'$, where $t = f(t_1, \dots, t_n)$, its *jungle representation* is the hyperedge replacement jungle rewrite rule $\mathcal{J}(t \rightarrow t') = f \rightsquigarrow (R, v_0, \dots, v_n)$ such that

- $R = \mathcal{J}(\{t', t_1, \dots, t_n\})$, i.e., R is the fully collapsed jungle representing the rhs and all the arguments of the lhs of the rule $t \rightarrow t'$.
- Nodes v_0, \dots, v_n are determined (see Proposition 1.12) by $\text{term}_R(v_0) = t'$ and $\text{term}_R(v_i) = t_i$ for $1 \leq i \leq n$.

The jungle representation of a rewrite rule just defined may be understood more easily by resorting to a “normalized” presentation of the rule itself. If $f(t_1, \dots, t_n) \rightarrow t'$ is a rule, its “normal” form is

$$f(x_1, \dots, x_n) \rightarrow t', \text{ where } x_1 = t_1, \dots, x_n = t_n,$$

with x_1, \dots, x_n fresh, distinct variables. The jungle representation $(L \xleftarrow{l} K \xrightarrow{r} R)$ of the rule can be considered as a direct translation of this normal form, since L represents exactly $f(x_1, \dots, x_n)$, R represents the term t' and the arguments of the lhs t_1, \dots, t_n , and morphism r forces the identifications $x_i = t_i$ of the *where* clause. Clearly, such a normalized presentation is equivalent to the original one (as proved below by the soundness and completeness results), since the pushout in the category of jungles corresponds to term unification.

Example 3.3 (*A jungle rewrite rule*). Let $\underline{\Sigma}$ be the signature of lists of natural numbers introduced in Example 1.4. Then the term rewrite rule $\text{car}(\text{cons}(x, y)) \rightarrow x$ is represented by the following jungle rewrite rule (Fig. 11), where we use the conventions of Definition 2.1. This representation should be compared with the normal form of such a rule, which is

$$\text{car}(x_1) \rightarrow x, \text{ where } x_1 = \text{cons}(x, y).$$

This example shows that also rewrite rules with a single variable on the right-hand side are allowed.

Definition 3.4 (*The jungle-rewriting system representing a TRS*). If $\mathcal{T} = \{t_i \rightarrow t'_i\}_{i \leq n}$ is a term-rewriting system, its *jungle representation* is the hyperedge replacement jungle-rewriting system $\mathcal{J}(\mathcal{T}) = \{\mathcal{J}(t_i \rightarrow t'_i)\}_{i \leq n}$.

3.2. Soundness and completeness of the jungle representation

The first result of this section states the soundness of our representation of term rewrite rules, i.e., that every rewriting performed with the jungle representation of a term rule corresponds to one or more applications of that rule. We first present a fundamental lemma that considers the case where the outermost operator of a term is rewritten in a direct jungle rewriting: this is shown to be equivalent to a single application of the rewrite rule. Then we state the soundness theorem, which easily follows from the lemma by induction on the structure of the jungle. The proof of this second part is borrowed from [22].

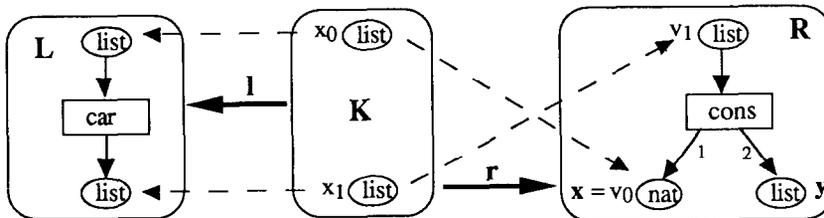


Fig. 11.

Lemma 3.5 (Soundness of topmost rewriting). *Let $p = L \stackrel{\perp}{\leftarrow} K \stackrel{\rightarrow}{\rightarrow} R$ be the jungle representation of the term rewrite rule $t \rightarrow t'$ (i.e., $p = \mathcal{J}(t \rightarrow t')$), and let $G \Rightarrow_p H$ be a direct rewriting based on $g: L \rightarrow G$ as in Fig. 9, with $\text{Var}(G) = \emptyset$. Then if $v \in G_V$ is the image through g of the (unique) root node of L , i.e., $g_V(\text{ROOT}_L) = \{v\}$, then $\text{term}_G(v)$ rewrites to $\text{term}_H(\text{tr}(v))$ using rule $t \rightarrow t'$, where $\text{tr}: V_G \rightarrow V_H$ is the track function of Definition 2.4.*

Proof. We show that there exists a substitution σ such that $\text{term}_G(v) = \sigma(t)$, and $\text{term}_H(\text{tr}(v)) = \sigma(t')$. Consider indeed Fig. 12 obtained from Fig. 9 by decorating each jungle with the set of terms it represents and each morphism with its induced substitution, and assuming that $t = f(t_1, \dots, t_n)$ (some redundant information is omitted):

$$\begin{array}{ccccc}
 \{f(x_1, \dots, x_n)\} & \xleftarrow[1]{\{x_0/f(x_1, \dots, x_n)\}} & \{x_0, x_1, \dots, x_n\} & \xrightarrow[r]{\{x_0/t', x_i/t_i\}} & \{t', t_1, \dots, t_n\} \\
 \{x_i/s_i\} \downarrow g & & \downarrow k \{x_0/y, x_i/s_i\} & & \downarrow h \sigma \\
 \{f(s_1, \dots, s_n), \dots\} & \xleftarrow[d]{\{y/f(s_1, \dots, s_n)\}} & \{y, s_1, \dots, s_n, \dots\} & \xrightarrow[b]{\{y/\sigma(t')\}} & \{\sigma(t'), s_1, \dots, s_n, \dots\} \\
 G & & D & & H
 \end{array}$$

Fig. 12.

The top row of Fig. 12 is the jungle representation of the rule $f(t_1, \dots, t_n) \rightarrow t'$. Clearly, if v is the image through g of the root node of L , $\text{term}_G(v)$ must be of the form $f(s_1, \dots, s_n)$, for suitable s_1, \dots, s_n . Thus, the terms represented by G must include $f(s_1, \dots, s_n)$, possibly among others. By Proposition 1.19 and Assumption 2.3, D is obtained from G by removing the edge in the image of g : its source node becomes a variable (denoted here by y), and it is the unique variable of D since G is ground by hypothesis (thus, the substitution induced by $d: D \rightarrow G$ is necessarily $\{y/f(s_1, \dots, s_n)\}$). To make the left square commutative, k must map x_0 to y and x_i to the root of s_i for each $1 \leq i \leq n$. Next, by Proposition 1.18, since the right square is a pushout, the substitutions $\langle \sigma_b, \sigma_h \rangle$ are the most general unifiers of $\langle \sigma_k = \{x_0/y, x_i/s_i\}, \sigma_r = \{x_0/t', x_i/t_i\} \rangle$; denoting σ_h by σ , this implies that $\sigma_b(y) = \sigma(t')$ and $\sigma_b(s_i) = \sigma(t_i)$ for each $1 \leq i \leq n$. But since G is acyclic, all s_i must be ground and, thus, $\sigma_b(s_i) = s_i$. Thus, we have $\text{term}_G(v) = f(s_1, \dots, s_n) = f(\sigma(t_1), \dots, \sigma(t_n)) = \sigma(f(t_1, \dots, t_n))$, and $\text{term}_H(\text{tr}(v)) = \text{term}_H(h_V \circ d_V^{-1}(v)) = \text{term}_H(h_V(y)) = \sigma(t')$. \square

Theorem 3.6 (Soundness of jungle representation of rewrite rules). *Let $p = L \stackrel{\perp}{\leftarrow} K \stackrel{\rightarrow}{\rightarrow} R$ be the jungle representation of the term rewrite rule $t \rightarrow t'$ (i.e., $p = \mathcal{J}(t \rightarrow t')$), and let $G \Rightarrow_p H$ be a direct rewriting as in Fig. 9, with $\text{Var}(G) = \emptyset$. Then for each $v \in V_G$ $\text{term}_G(v)$ rewrites to $\text{term}_H(\text{tr}(v))$ by n applications of the rule $t \rightarrow t'$, where n is the number of distinct paths from v to the image through g of the (unique) root node of L ; call it \underline{v} .*

Proof. By bottom-up induction over V_G . Since G has no variables, every node of G is the source of some edge. Suppose that the statement is true for all the target nodes of an edge $e \in E_G$. Then either the source node of e is \underline{v} itself, and in this case the statement follows directly by Lemma 3.5 (because there is exactly one path, the empty one, from \underline{v} to \underline{v}), or it is not. In the last case, let $s_G(e) = v$ and $t_G(e) = v_1 \dots v_m$ (with $m \geq 0$). Then clearly the number of distinct paths from v to \underline{v} is $n = n_1 + \dots + n_m$, where n_i is the number of paths from v_i to \underline{v} . Since $term_G(v) = m_G(e)(term_G(v_1), \dots, term_G(v_m))$ and by induction hypothesis $term_G(v_i)$ rewrites to $term_H(tr(v_i))$ in n_i steps, then $term_G(v)$ rewrites in $n_1 + \dots + n_m = n$ steps to $m_G(e)(term_H(tr(v_1)), \dots, term_H(tr(v_m)))$. On the other hand, by the explicit construction of the pushout complement in Proposition 1.9 and by Assumption 2.3, since D is obtained from G by removing only the edge having as source \underline{v} , there is a unique edge e' in E_D such that $d_E(e') = e$ (and, thus, $m_D(e') = m_G(e)$). Therefore, we have $m_G(e)(term_H(tr(v_1)), \dots, term_H(tr(v_m))) = m_D(e')(term_H(tr(v_1)), \dots, term_H(tr(v_m))) = m_H(b_E(e'))(term_H(tr(v_1)), \dots, term_H(tr(v_m))) = term_H(tr(v))$, because $tr(v)$ is the source node of $b_E(e')$. \square

The next result states that whenever a term rewrite rule $t \rightarrow t'$ can be applied to a term s , then its jungle representation $\mathcal{J}(t \rightarrow t')$ can be applied to *any* jungle which represents s (possibly among other terms). This result does not hold (without additional conditions) for non-left-linear rules in all the related works on graph term rewriting we are aware of. The reason is that in all these proposals the lhs L of the graph rewrite rule representing a term rewrite rule $t \rightarrow t'$ (call it $\mathcal{G}(t \rightarrow t')$) is supposed to represent the whole term t . Now if $t = f(x, x)$, rule $t \rightarrow t'$ can be applied to the ground term $f(a, a)$ via the substitution $\sigma = \{x/a\}$, but there is at least one graph representation of $f(a, a)$, say G , for which there is no direct rewriting via $\mathcal{G}(t \rightarrow t')$, namely, the one where the two occurrences of the constant a are represented by distinct edges. In fact, as shown in Fig. 13, there is no graph morphism from L to G (this observation holds both for jungles and for dags).

In our representation, just the topmost operator of t is represented in L , with pairwise distinct variables as arguments. In practice, we use a left-linear representation of (possibly non-left-linear) rules, and this is possible only thanks to the fundamental choice of performing all the constructions in the category of jungles.

Proposition 3.7 (Completeness w.r.t. applicability). *Let $\mathcal{J}(t \rightarrow t')$ be the jungle representation of a term rewrite rule $t \rightarrow t'$, G be a jungle (without variables) and $s \in TERM(G)$.*

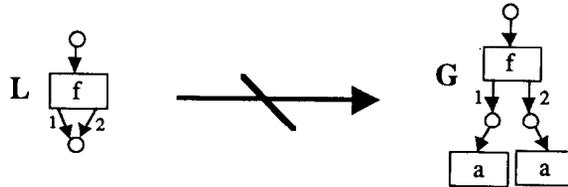


Fig. 13.

If $t \rightarrow t'$ can be applied to term s , then there is a direct rewriting $G \Rightarrow_{\mathcal{J}(t \rightarrow t')} H$ for some jungle H .

Proof. In order to prove the existence of a direct rewriting, we have to find a morphism $g: L \rightarrow G$, and we must show that after the construction of the pushout complement, the rhs pushout exists. Let $\mathcal{J}(t \rightarrow t') = L \xleftarrow{l} K \xrightarrow{r} R$. By hypothesis, there is a substitution σ such that $\sigma(t) = s$. Without loss of generality, we can suppose that $t = f(t_1, \dots, t_n)$ and $s = f(s_1, \dots, s_n)$, with $s_i = \sigma(t_i)$ for all $1 \leq i \leq n$.

Let v be any node of G such that $\text{term}_G(v) = s$, and let $g: L \rightarrow G$ be the morphism determined by $g_V(x_0) = v$ (by Fact 1.6), where x_0 is the unique root node of L . The fact that g is indeed a morphism can be checked easily by observing that all the target nodes of the unique edge of L are distinct. Next, the pushout complement object D can be constructed in the usual way (by Proposition 1.19, using Assumption 2.3). The resulting situation is exactly as in the diagram of Lemma 3.5: by Proposition 1.18 the rhs pushout exists if and only if the pair of substitutions $\langle \{x_0/t', x_1/t_1, \dots, x_n/t_n\}, \{x_0/y, x_1/s_1, \dots, x_n/s_n\} \rangle$ has a unifier, i.e., a pair $\langle \theta, \theta' \rangle$ of substitutions such that $\theta \circ \{x_0/t', x_1/t_1, \dots, x_n/t_n\} = \theta' \circ \{x_0/y, x_1/s_1, \dots, x_n/s_n\}$. But the pair $\langle \sigma, \{y/\sigma(t')\} \rangle$ clearly satisfies this requirement. \square

Note that, under the hypotheses of Proposition 3.7, if s' is the result of the application of the rule $t \rightarrow t'$ to s , then in general it is not true that $s' \in \text{TERM}(H)$. This is due to the fact that the single jungle rewriting step $G \Rightarrow_{\mathcal{J}(t \rightarrow t')} H$ may correspond to many term-rewriting steps, as stated in Theorem 3.6.

The following example shows a successful direct derivation in a paradigmatic situation where other approaches fail.

Example 3.8 (*Application of a non-left-linear rule*). Consider the rule (over the signature of Example 1.4) $\text{rem}(x, \text{cons}(x, y)) \rightarrow \text{rem}(x, y)$, which is intended to model the deletion of the occurrences of a given number from a list. It is not left-linear. Clearly, the ground term $\text{rem}(0, \text{cons}(0, \text{EMPTY}))$ rewrites to $\text{rem}(0, \text{EMPTY})$ via that rule. Figure 14 shows that the jungle representation of the rule can be applied also to the jungle representing $\text{rem}(0, \text{cons}(0, \text{EMPTY}))$, where the two occurrences of “0” are represented by distinct edges.

It must be stressed that as the result of the rewriting, the two distinct edges labeled by the constant “0” are merged in the resulting jungle H . Note that the two edges were not in the image of g : as pointed out in Section 2.2, the application of a hyper-edge replacement jungle rule can have nonlocal effects (see also the example after Lemma 1.17).

3.3. From HR jungle-rewriting systems to TRSs

The correspondence between term-rewriting systems and hyperedge replacement jungle-rewriting systems can be explored in the other direction as well, a topic which

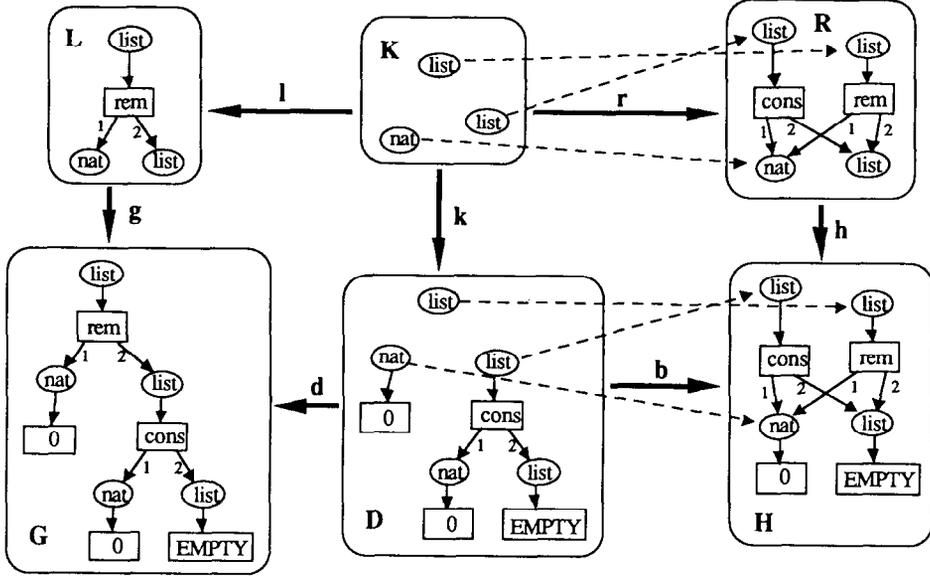


Fig. 14.

has never been considered in the related literature, as far as we know (a hint in this direction can be found in [30]). Indeed, it is possible to extract from every hyperedge replacement jungle-rewriting system \mathcal{R} a term-rewriting system $\mathcal{T}(\mathcal{R})$ such that \mathcal{R} is a sound representation of $\mathcal{T}(\mathcal{R})$. This result shows that the expressive power of hyperedge replacement jungle rewriting should not be much greater than that of TRSs, as far as ground rewriting is concerned. A similar result for nonground rewriting (but for a restricted kind of signatures) is presented in Section 4. The importance of these results for establishing an upper bound to the expressive power of hyperedge replacement jungle rewriting is commented on briefly in Section 6.

Definition 3.9 (*The TRS associated with a HR jungle-rewriting system*). Let $p = f \rightsquigarrow (R, v_0, \dots, v_n)$ be a HR jungle rewrite rule over \mathbf{Jungle}_{Σ} . The term rewrite rule associated with p , $\mathcal{T}(p)$, is defined as

$$f(t_1, \dots, t_m) \rightarrow t',$$

where $t' = \text{term}_R(v_0)$, and for each $1 \leq i \leq n$, $t_i = \text{term}_R(v_i)$. If $\mathcal{R} = \{p_i\}_{i \leq n}$ is a HR jungle rewriting system, then its associated TRS is defined as $\mathcal{T}(\mathcal{R}) = \{\mathcal{T}(p_i)\}_{i \leq n}$.

It must be stressed that in passing from a jungle rewrite rule to the associated term rewrite rule some information might get lost. More precisely, by Definition 3.9 it follows that if there is a node $v \in V_R$ such that $\text{term}_R(v)$ does not occur as a subterm in $\{t', t_1, \dots, t_m\}$ then $\text{term}_R(v)$ does not appear in the associated rewrite rule. Those terms should be considered as “garbage” if we just consider the term rule associated with the jungle rule. Nevertheless, the following result holds.

Proposition 3.10 (\mathcal{R} is sound with respect to the associated TRS). *Let $\mathcal{R} = \{p_i\}_{i \leq n}$ and $\mathcal{T}(\mathcal{R}) = \{\mathcal{T}(p_i)\}_{i \leq n}$ be as Definition 3.9. Then*

- (1) *For each jungle G with $\text{Var}(G) = \emptyset$, if $G \Rightarrow_{p_i} H$ based on $g: L \rightarrow G$, then for each $v \in V_G$ $\text{term}_G(v)$ rewrites to $\text{term}_H(\text{tr}(v))$ by n applications of the rule $t \rightarrow t'$, where n is the number of paths from v to the image through g of the unique root node of L (see Fig. 9).*
- (2) *If $s \rightarrow_{\mathcal{T}(\mathcal{R})} s'$ using rule $\mathcal{T}(p_i)$, then for each jungle G such that $s \in \text{TERM}(G)$ there exists a jungle H such that $G \Rightarrow_{\mathcal{T}(p_i)} H$.*

Proof (outline). The proposition states that \mathcal{R} enjoys the same properties of soundness and completeness of applicability w.r.t. $\mathcal{T}(\mathcal{R})$ as its jungle representation $\mathcal{J}(\mathcal{T}(\mathcal{R}))$. This can be checked for each jungle rewrite rule in \mathcal{R} by constructing a double pushout diagram which results to be almost identical to Fig. 12, except for the fact that the rhs jungle R may represent other terms besides $\{t', t_1, \dots, t_n\}$. It is easy to verify that those additional terms do not influence the existence of the pushout. Then the proofs of Lemma 3.5, Theorem 3.6, and Proposition 3.7 work in this case as well. \square

The results of soundness of our jungle representation of TRSs and of completeness w.r.t. applicability could allow us to restate in our framework most of the results about graph representation of term-rewriting systems, like, for example, the completeness for confluent and terminating TRSs [21, 22]. However, this goes beyond the scope of this paper and is left as a topic for future research.

4. Logic programming

The main notion of the operational semantics of logic programming, i.e., the *resolution step*, can be regarded as a special kind of term rewriting. There are three main differences between a resolution step and an ordinary term-rewriting step. First, the resolution involves full unification instead of pattern matching; second, a “state” of a logic program is a collection of terms (actually, atomic formulas) instead of a single term; and third, in a resolution step the rewriting is always performed at the top level, because the clauses of a program state how atomic formulas can be rewritten and the formulas cannot be nested. In spite of the similarity between rewriting and resolution steps, only recently the representation of logic programming by graph rewriting has been explored by the authors in joint works with others [5, 7]. We believe that the fact that none of the many approaches to term graph rewriting has been extended to cover logic programming as well can be motivated by the impossibility of modeling the unification in such frameworks (indeed, the other two differences mentioned above can be handled easily also in other graph term-rewriting approaches). On the contrary, as we show in this section, using jungle rewriting and exploiting the correspondence between pushouts and term unification, logic programs can be represented easily

by hyperedge replacement jungle grammars, in a way that is not only sound but also complete.

This section is essentially a reorganization of the results appeared in [5]. Here they are seen as an instantiation of the general framework of hyperedge replacement jungle rewriting presented in Sections 1 and 2, while in [5] they were described using an ad hoc representation of formulas (i.e., a mixture of dags and jungles), and the relationship with term-rewriting systems was not stressed. After introducing the basic definitions about the operational semantics of logic programs, we show in Section 4.2 how to represent goals (i.e., multisets of atomic formulas) by jungles. In Section 4.3 we propose a jungle representation of program clauses, and we report the main result which states the soundness and completeness of this representation. Finally, in Section 4.4 we extend the representation to programs, by associating a jungle grammar with each pair (logic program, goal). In this section we also discuss how to extract a logic program $\mathcal{P}(\mathcal{G})$ from a given jungle grammar \mathcal{G} , and explore the relationship between \mathcal{G} and $\mathcal{P}(\mathcal{G})$.

4.1. Background

For an introductory and complete treatment of logic programs, see [26]. Here we introduce just the concepts which will be used in the rest of the paper. The basic ingredients of a logic program are atomic formulas, which are simply terms built over a two-sorted signature (including predicate and function symbols) satisfying the restriction that predicate symbols are always at the top of a term.

Definition 4.1 (*Logic programming signature*). A logic programming signature is a signature $\underline{\Sigma} = (S, \Sigma)$ such that

- S contains exactly two elements, which, by convention, will be denoted by s and b , respectively, i.e., $S = \{s, b\}$. Sort s will be called the *sort of terms*, and b (for “boolean”) will be called the *sort of predicates*.
- Each $\Sigma_{w,x}$ is empty if $b \in w$, i.e., the sort of predicates does not appear in the arity of any operator.

Thanks to these restrictions, a logic programming signature $\underline{\Sigma}$ will be equivalently represented as $\underline{\Sigma} = (\Pi, \Phi)$, where $\Pi = \bigcup_n \Pi_n$ is the ranked set of predicate symbols and $\Phi = \bigcup_n \Phi_n$ is the ranked set of function symbols, with $\Pi_n \triangleq \Sigma_{s^n, b}$ and $\Phi_n \triangleq \Sigma_{s^n, s}$.

We stress that the treatment of logic programming presented in this section can be straightforwardly generalized to allow many-sorted terms, simply by requiring that there exists a special sort b satisfying the second condition above. We restrict ourselves to single-sorted terms in order to keep ourselves closer to the tradition of logic programming.

Definition 4.2 (*Formulas, clauses, goals, logic programs*). Given a logic programming signature $\underline{\Sigma} = (\Pi, \Phi)$, an *atomic formula* over $\underline{\Sigma}$ is any term of sort b . A (*conjunctive*)

formula is a list of atomic formulas separated by commas, like B_1, \dots, B_n . A *definite clause* C is an expression of the form $C = (H :- B_1, \dots, B_n)$ with $n \geq 0$, where H, B_1, \dots, B_n are atomic formulas, “:-” represents logic implication (from right to left), and the comma represents logical conjunction. H is called the *head* of C and (B_1, \dots, B_n) its *body*. A *goal* A is an expression like $A = (- A_1, \dots, A_n)$ with $n \geq 0$, where each A_i is an atomic formula called an *atomic subgoal* of A . If $n = 0$, A is called the *empty goal*, and is denoted by \square . A *logic program* P is a finite set of definite clauses.

For the formal treatment of the various semantics of a logic program and their equivalence see [26]. In this paper, since we are interested in showing that the operational behavior of a program can be faithfully simulated by a hyperedge replacement jungle grammar, we will introduce just the basic notions related to the operational semantics.

Definition 4.3 (*Resolution steps, refutations, computed answer substitutions*). Given a clause $C = (H :- B_1, \dots, B_n)$ and a goal $A = (- A_1, \dots, A_n)$ with no common variables, A' is the *resolvent* of A and C via θ if

- there exists an atomic subgoal A_i of A and a substitution θ which is the most general unifier of A_i and H , and
- $A' = :-\theta(A_1, \dots, A_{i-1}, B_1, \dots, B_m, A_{i+1}, \dots, A_n)$.

In this case we will say that there is a *resolution step from A to A' via C and θ* , and we write $A \rightarrow_{C, \theta} A'$.

A *refutation* of a goal A is a finite sequence of resolution steps which starts with A and ends with the empty goal. If the refutation has length n , where step i uses clause C_i and the mgu θ_i , then the substitution $\theta = (\theta_n \circ \dots \circ \theta_1)|_{\text{var}(A)}$ (i.e., the restriction of $\theta_n \circ \dots \circ \theta_1$ to the variables appearing in A), is called a *computed answer substitution* for A . In this case we say that there is a *refutation of A via C_1, \dots, C_n and $\theta = (\theta_n \circ \dots \circ \theta_1)|_{\text{var}(A)}$* .

4.2. Representing formulas by jungles

Since atomic formulas are defined as terms of type b over a logic programming signature $\underline{\Sigma}$, the function *term* of Definition 1.9 will return an atomic formula when applied to a node labeled by b of a jungle over $\underline{\Sigma}$. However, for the representation of goals or formulas, which are lists of atomic formulas, it is not correct to use the function *TERM*, which returns a *set* of terms. Indeed, by representing a list as a set we would lose information about the multiplicity and the ordering of the atomic subformulas of a formula. On the other hand, at least the information about the multiplicity of the atomic subgoals in a goal is necessary if we want to represent faithfully the operational behavior of a program. For example, the goals $A = (-q(x))$ and $A' = (-q(x), q(x))$ may behave quite differently (i.e., the sets of their refutations can be different) with respect to the same program P , although they are logically equivalent.

Thus, we define below a function $FORM$, which associates with each jungle over a logic programming signature $\underline{\Sigma}$ the *multiset* of formulas it represents. Since the elements of a multiset are not ordered, we still lose the ordering of atomic subgoals in a goal. Thus, it is not possible to define a selection rule based on such ordering in the jungle-rewriting framework we are defining. Nevertheless, we believe that this is mainly a matter of implementation, because the operational semantics of a logic program does not depend on the selection rule (see [26]).

In the definition of function $FORM$ we also take into account the fact that “variables” of sort b (the sort of predicates) have no meaning in logic programming; thus, they are ignored as “garbage”. Note that in a jungle a variable of sort b must be an isolated node, because of the second requirement of Definition 4.1. From now on we will denote the category of jungles over a logic programming signature $\underline{\Sigma}=(\Pi, \Phi)$ by $\mathbf{Jungle}_{\Pi, \Phi}$.

Definition 4.4 (*From jungles to multisets of atomic formulas*). Let G be a jungle of $\mathbf{Jungle}_{\Pi, \Phi}$. The *multiset of atomic formulas* represented by G is defined as

$$FORM(G) = \{ \langle term_G(v), v \rangle \mid v \in V_G, m_G(v) = b, \text{ and } outdegree_G(v) \neq 0 \}.$$

We adopt the following conventions in the representation of multisets. A multiset M of elements of a set S is a set of pairs $M = \{ \langle s, x \rangle \}$, where $s \in S$ and x belongs to a set of tags used to distinguish among different occurrences of the same element. The usual representation of M as a function $f_M: S \rightarrow \mathbb{N}$ can be obtained by putting $f_M(s) = | \{ \langle x, y \rangle \in M \mid x = s \} |$. All the operations on multisets will not use the identity of tags.

For the multisets of formulas M and M' , we write $M \approx M'$ if $f_M = f_{M'}$, and $M \cong M'$ if $f_M = f_{M'}$ modulo a variable renaming.

In order to show how the clauses of a logic program can be represented by hyperedge replacement jungle rewrite rules, we first need to define when a jungle represents a collection of atomic formulas (or a goal) and, possibly, we have to characterize a distinguished jungle representation of a given goal. We present now a variant of Definition 1.11 and Proposition 1.12 (adapted from [5]) which takes into account the fact that we deal with multisets of atomic formulas, but still with sets of terms. For the sake of simplicity, in the sequel by *collection* we mean a *set* of terms, but a *multiset* of atomic formulas.

Definition 4.5 (*Representing formulas and terms by a jungle*). Let F be a multiset of atomic formulas and T be a set of terms of sort s over $\underline{\Sigma}=(\Pi, \Phi)$. Moreover, let \overline{T}_F be the set of all the terms of sort s which occur either in T or in F . In general, $T \subset \overline{T}_F$ because \overline{T}_F contains also all the arguments of the formulas in F and is closed under the subterm relation. Then a jungle G of $\mathbf{Jungle}_{\Pi, \Phi}$ *represents* (T, F) iff $FORM(G) \cong F$, and $TERM_s(G) \cong \overline{T}_F$. The *collection of terms and formulas associated with* G , \mathcal{F}_G , is defined as $\mathcal{F}_G = (TERM_s(G), FORM(G))$, while the *goal associated with* G is simply $FORM(G)$.

For a given collection of terms and formulas, a final representation as a jungle (in the sense of Proposition 1.12) does not always exist. Nevertheless, a unique jungle (up to isomorphisms) can be characterized by a slightly weaker condition, if we restrict ourselves to morphisms which are injective when restricted to nonvariable nodes of sort b . These morphisms have the property of preserving the multiplicity of formulas.

Proposition 4.6 (The collapsed jungle representing a collection of terms and formulas). *Let (T, F) be a collection of terms and formulas, as in Definition 4.5. Then the set of jungles representing (T, F) includes an element $\mathcal{J}(T, F)$ (determined up to isomorphisms), such that for all G which represent (T, F) there exists a jungle morphism $h_G: G \rightarrow \mathcal{J}(T, F)$ which is injective on nonvariable nodes of sort b , and maps variable nodes of that sort to variable nodes. This morphism is unique up to an arbitrary permutation of the nodes which are the roots of distinct occurrences of the same atomic formula.*

Proof. Let F, T , and \bar{T}_F be as in Definition 4.5, and let $\mathcal{J}(\bar{T}_F)$ be the fully collapsed jungle representing \bar{T}_F as defined in Proposition 1.12. Then for each element $\langle t, x \rangle$ of the multiset F add to $\mathcal{J}(\bar{T}_F)$ one node $v_{t,x}$ labeled by b , and one edge $e_{t,x}$ labeled by the predicate symbol of t . Next put $s(e_{t,x}) = v_{t,x}$, and if $t = q(t_1, \dots, t_n)$ then put $t(e_{t,x}) = v_1 \dots v_n$, where v_i is the unique node of $\mathcal{J}(\bar{T}_F)$ such that $\text{term}(v_i) = t_i$. Finally, add a single isolated node \underline{v} labeled by b and call the resulting jungle $\mathcal{J}(T, F)$.

By construction, $\mathcal{J}(T, F)$ represents (T, F) . Furthermore, if G is a jungle which represents (T, F) , a morphism $h_G: G \rightarrow \mathcal{J}(T, F)$ of the type described above is uniquely determined on nodes of sort s , by Proposition 1.12, it must map variable nodes of sort b to \underline{v} , and on nonvariable nodes of sort b it can be forced to be one-to-one, because the number of such edges in G and $\mathcal{J}(T, F)$ is identical. \square

4.3. Representing clauses as jungle rewrite rules

After understanding how to represent a collection of terms or atomic formulas as jungles, we show how a program clause can be represented by a hyperedge replacement jungle rewrite rule.

Definition 4.7 (The HR jungle rule representing a program clause). Let $C = (q(t_1, \dots, t_m) :- B_1, \dots, B_n)$ be a program clause. Then its *jungle representation* is the hyperedge replacement jungle rewrite rule $\mathcal{J}(C) = q \rightsquigarrow (R, v_0, \dots, v_m)$ such that

- $R = \mathcal{J}(\{t_1, \dots, t_n\}, (B_1, \dots, B_n))$, i.e., it is the fully collapsed jungle representing the set of terms including all the arguments of the head of C , and the collection of atomic formulas appearing in the body of C .
- Nodes v_0, \dots, v_m are such that $v_0 = \underline{v}$ (the unique variable node of sort b in R , by Proposition 4.6), and for $1 \leq i \leq m$ v_i is the unique node such that $\text{term}_R(v_i) = t_i$, because term_R is injective on nodes labeled by s by the construction in Proposition 4.6.

Example 4.8 (Representing a clause with a HR jungle rewrite rule). Consider the clause

$$C = \text{reverse}(\text{cons}(x, y), z) :- \text{reverse}(y, w), \text{append}(w, \text{cons}(x, \text{nil}), z)$$

over the logic programming signature $\underline{\Sigma} = (\{\Pi_2 = \{\text{reverse}\}, \Pi_3 = \{\text{append}\}\}, \{\Phi_0 = \{\text{nil}\}, \Phi_2 = \{\text{cons}\}\})$. The jungle representation of C , $\mathcal{J}(C)$, is shown in Fig. 15, where $r_V(x_i) = v_i$ for $0 \leq i \leq 2$.

As for the case of term-rewriting rules discussed in Section 3.1, also the jungle representation of a clause C can be understood more easily by considering a “normalized” presentation of the clause, which has been called its *canonical form* in [5]. The canonical form of C , $\text{can}(C)$, is a clause equivalent to C (in the sense that for a given goal A , either the application to A of both clauses fails, or it succeeds producing the same substitution for the variables in A), such that the argument terms of the head are replaced by fresh distinct variables, and the body is enriched with equalities forcing the unification of the new variables with the corresponding terms. For example, if C is the clause of Example 4.8, then

$$\begin{aligned} \text{can}(C) \triangleq & \text{reverse}(x_1, x_2) :- x_1 = \text{cons}(x, y), x_2 = z, \\ & \text{reverse}(y, w), \text{append}(w, \text{cons}(x, \text{nil}), z). \end{aligned}$$

The jungle representation of C can then be considered as a direct representation of $\text{can}(C)$, since L represents its head, R represents its body, and $r : K \rightarrow R$ forces the added equalities to hold.

As for the term-rewriting systems, in the case of logic programs also it is possible to go the other way around, i.e., it is possible to extract a program clause C from a hyperedge replacement jungle rewrite rule $q \rightsquigarrow (R, v_0, \dots, v_n)$ over a logic programming signature $\underline{\Sigma} = (\Pi, \Phi)$, provided that $q \in \Pi$.

Definition 4.9 (The clause associated with a HR jungle rewrite rule). Let $p = q \rightsquigarrow (R, v_0, \dots, v_n)$ be a hyperedge replacement jungle rewrite rule over a logic programming signature $\underline{\Sigma} = (\Pi, \Phi)$, such that $q \in \Pi$. Then the *clause associated with p* is

$$\mathcal{C}(p) = q(t_1, \dots, t_m) :- B_1, \dots, B_n,$$

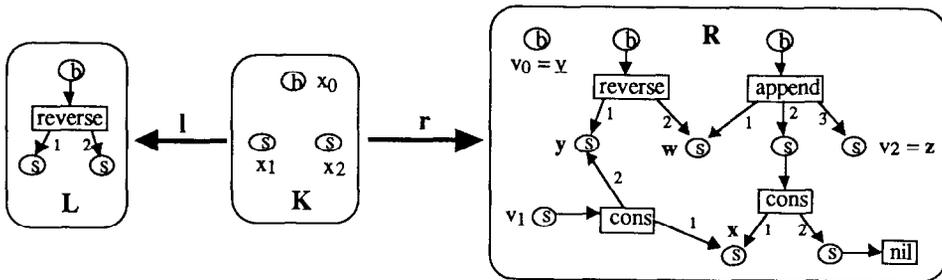


Fig. 15.

where $(B_1, \dots, B_n) = \text{FORM}(R)$, and for each $1 \leq i \leq m$, $t_i = \text{term}_R(v_i)$.

Also in this case (as when extracting a term rewrite rule from a jungle rule) some information might get lost, in the sense that some term represented by R could not appear in the associated clause. Moreover, it must be stressed that $\mathcal{C}(p)$ is defined up to the ordering of the atomic formulas in its body.

We will show now that the representation of a program clause C by a HR jungle rewrite rule is not only sound but also complete. The results will be extended next to entire programs, showing the existence of a close relation between logic programs and HR jungle grammars over a logic programming signature. In order to state formally these results, we will use the substitutions associated with direct derivations and with derivations, as introduced in Definitions 2.4 and 2.7: they will play a role similar to the track functions for the case of term-rewriting systems.

Informally, by soundness we mean that if there is a direct derivation from a jungle G to a jungle H via rewrite rule $\mathcal{J}(C)$ and induced substitution σ , then $\text{FORM}(G) \rightarrow_{C, \sigma} \text{FORM}(H)$, i.e., there is a resolution step from goal $\text{FORM}(G)$ to goal $\text{FORM}(H)$ via clause C and σ . Conversely, by completeness we mean that if $A \rightarrow_{C, \sigma} B$, then for any jungle G which represents A the rewrite rule $\mathcal{J}(C)$ can be applied to G with induced substitution σ , and the derived jungle H represents B . This result of completeness (proved below) is stronger than the corresponding one for TRSs (Proposition 3.7), and holds essentially because rewriting is always performed at the top level of a goal in a logic program computation. Indeed, if just topmost rewriting is allowed, then a similar result of completeness can easily be shown to hold for TRSs as well.

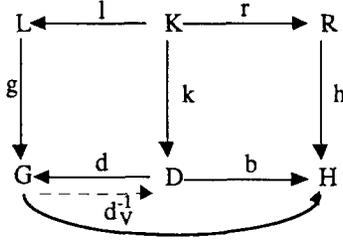
Theorem 4.10 (Soundness and completeness of jungle representation of clauses). *Let $C = q(t_1, \dots, t_m) :- B_1, \dots, B_n$ be a definite clause and $\mathcal{J}(C) = L \xleftarrow{l} K \xrightarrow{r} R$ be its jungle representation, as described in Definition 4.7. Then:*

Soundness: If G is a jungle, $g: L \rightarrow G$ is an occurrence of L in G , and $G \Rightarrow_{\mathcal{J}(C)} H$ with associated substitution σ_{tr} (see Definition 2.4 or Fig. 16) then $\text{FORM}(G) \rightarrow_{C, \theta} \text{FORM}(H)$ (see Definition 4.3), with $\theta = (\sigma_h \cup \sigma_{tr})_s$, i.e., the restriction of $\sigma_h \cup \sigma_{tr}$ to variables of sort s . Moreover, σ_{tr_s} is the restriction of θ to the variables in $\text{FORM}(G)$.

Completeness: If $A \rightarrow_{C, \theta} B$, then for every jungle G which represents A there exists an occurrence $g: L \rightarrow G$ such that $G \Rightarrow_{\mathcal{J}(C)} H$, H represents B , $\theta \cong (\sigma_h \cup \sigma_{tr})_s$, and $\theta|_{\text{Var}(A)} = \sigma_{tr_s}$.

Proof. For simplicity and without loss of generality, we assume that in Fig. 16 the variables of R and D are disjoint, while the nodes of G and D are the same (i.e., $V_G = V_D$) and d_V is the identity function. As a consequence, we have $\sigma_{tr_s} = \sigma_{b_s}$.

Soundness: Suppose that $G \Rightarrow_{\mathcal{J}(C)} H$, and $\text{term}_G(\hat{v}) = q(s_1, \dots, s_m)$, where $\hat{v} = g_V(v)$ is the image of the unique root node of L . Then, the pushout complement object D exists by hypothesis and, by Proposition 1.19, it represents the collection of terms and

Fig. 16. Substitution: $\sigma_r: \text{Var}(G) \leftarrow T_{\Sigma}(\text{Var}(H))$.

formulas $(\text{TERM}_s(G) \cup \{s_1, \dots, s_m\}, \text{FORM}(G) - \{\langle q(s_1, \dots, s_m), \hat{v} \rangle\})$. By Definition 4.7, morphism r has the associated substitution $\sigma_r = \{x_0/\underline{v}, x_1/t_1, \dots, x_m/t_m\}$ (\underline{v} is isolated), while by construction k is associated with $\sigma_k = \{x_0/y, x_1/s_1, \dots, x_m/s_m\}$, with $y = d_v^{-1}(\hat{v}) = \hat{v}$. By Proposition 1.18, since the right square is a pushout by hypothesis, the substitutions $\langle \sigma_r, \sigma_k \rangle$ unify with the most general unifier $\langle \sigma_h, \sigma_b \rangle$; thus, for each $1 \leq i \leq m$ $\sigma_h(t_i) = \sigma_b(s_i)$. Therefore, C can be applied to $\text{FORM}(G)$ because its head $q(t_1, \dots, t_m)$ unifies with $q(s_1, \dots, s_m)$. Moreover, $\theta \triangleq \langle \sigma_h, \sigma_b \rangle_s = (\sigma_h \cup \sigma_r)_s$, by the above assumptions) is actually an mgu of $\langle t_1, \dots, t_m \rangle$ and $\langle s_1, \dots, s_m \rangle$, because node $x_0 \in V_K$ is mapped by both k and r to isolated nodes, and, thus, it does not influence the unification. Clearly, $\theta|_{\text{Var}(\text{FORM}(G))} = \sigma_{r_s}$. Finally, we have to show that H represents the resolvent of C and $\text{FORM}(G)$ via $\theta = (\sigma_h \cup \sigma_r)_s$, i.e., that $\text{FORM}(H) = \theta(\text{FORM}(G) - \{\langle q(s_1, \dots, s_m), \hat{v} \rangle\} \cup \{B_1, \dots, B_n\})$, as in Definition 4.3. Indeed, by Proposition 1.18, $\text{FORM}(H) = \sigma_b(\text{FORM}(D)) \cup \sigma_h(\text{FORM}(R)) = \sigma_b(\text{FORM}(G) - \{\langle q(s_1, \dots, s_m), \hat{v} \rangle\}) \cup \sigma_h(B_1, \dots, B_n) = \theta(\text{FORM}(G) - \{\langle q(s_1, \dots, s_m), \hat{v} \rangle\} \cup \{B_1, \dots, B_n\})$. Note that the restriction of the substitutions σ_b and σ_h to sort s is sound because there are no variables of type b in $\text{FORM}(A)$ for any A .

Completeness: If $A \rightarrow_{C, \theta} B$, G is a jungle such that $\text{FORM}(G) \cong A$, and $q(s_1, \dots, s_m)$ is the atomic subgoal of A rewritten by the application of C ; let $g: L \rightarrow G$ be defined in such a way that $\text{term}_G(g_V(v)) = q(s_1, \dots, s_m)$, for the unique root node v of L . Note that morphism g can be defined because the nodes in the target of the unique edge of L are all distinct. By Definition 4.3, we have $B \approx \theta(A - \{\langle q(s_1, \dots, s_m), g_V(v) \rangle\} \cup \{B_1, \dots, B_n\})$.

By Proposition 1.19, the pushout complement of $\langle l, g \rangle$ exists, and it is a jungle D which represents the collection of terms and formulas $(\text{TERM}_s(G) \cup \{s_1, \dots, s_m\}, A - \{\langle q(s_1, \dots, s_m), g_V(v) \rangle\})$. As above, we have that $\sigma_r = \{x_0/\underline{v}, x_1/t_1, \dots, x_m/t_m\}$ (\underline{v} is isolated), while $\sigma_k = \{x_0/y, x_1/s_1, \dots, x_m/s_m\}$, with $y = d_v^{-1}(g_V(v)) = g_V(v)$. By hypothesis, the variables of C and A are disjoint, and $\theta'(t_i) = \theta''(s_i)$, where $\theta' = \theta|_{\text{Var}(C)}$ and $\theta'' = \theta|_{\text{Var}(A)}$. Thus, $\langle \sigma_r, \sigma_k \rangle$ have a pair of unifying substitutions, namely, $\langle \theta' \cup \{v/v'\}, \theta'' \cup \{y/v'\} \rangle$, where v' is a fresh “variable” of type b .

Therefore, by Proposition 1.18, the pushout $\langle H, h: R \rightarrow H, b: D \rightarrow H \rangle$ of $\langle r, k \rangle$ exists, and it can be shown that $\sigma_h \cong \theta' \cup \{v/v'\}$ and $\sigma_b \cong \theta'' \cup \{y/v'\}$. Thus, $\theta \cong (\sigma_h \cup \sigma_b)_s = (\sigma_h \cup \sigma_r)_s$ by the assumption, and $\theta'' = (\sigma_r)$. Finally (again by Proposition 1.18),

graph H represents the goal $\sigma_b(FORM_D) \cup \sigma_h(FORM_R) = \theta''(A - \{\langle q(s_1, \dots, s_m), g_V(v) \rangle\}) \cup \theta'(B_1, \dots, B_n) = \theta(A - \{\langle q(s_1, \dots, s_m), g_V(v) \rangle\}) \cup \{B_1, \dots, B_n\}$, which is equal to B up to variable renaming. \square

4.4. Logic programs as HR jungle grammars and vice versa

We analyze now the correspondence between logic programs (over a fixed logic programming signature $\underline{\Sigma}$) and the class of hyperedge replacement jungle grammars over the same signature. For a given logic program P and a fixed goal A we define a HR jungle grammar $\mathcal{J}(P, A)$ such that each derivation in $\mathcal{J}(P, A)$ exactly corresponds to a refutation for A and vice versa. On the other hand, with each HR jungle grammar \mathcal{G} we associate a logic program $\mathcal{P}(\mathcal{G})$ and a goal A such that, again, refutations for A in $\mathcal{P}(\mathcal{G})$ and derivations of \mathcal{G} are in one-to-one correspondence.

To do this we first need to extend the correspondence between a resolution step and a direct derivation, stated in Theorem 4.10, to entire computations. The point is that, from the computational point of view, the relevant aspect of a refutation is that it returns a computed answer substitution, i.e., the substitution (restricted to the variables of the initial goal) obtained as the composition of the substitutions computed at each resolution step. We will show that the computed answer substitution of a refutation faithfully corresponds to the substitution associated with a derivation (see Definition 2.7).

In order to define a jungle grammar, by Definition 2.5, we have to choose a set of terminal colors. For a given logic programming signature $\underline{\Sigma} = (\Pi, \Phi)$, it is reasonable to take the subsignature $\underline{\Phi} \triangleq (\{s, b\}, \Phi)$ as the terminal colors. This implies that a rewriting $G \Rightarrow^* H$ is a derivation iff H does not include predicate symbols, which is exactly what is expected, since in this case H represents the empty goal, i.e., $FORM(H) = \square$.

Definition 4.11 (*The jungle grammar associated with a logic program and a goal*). Let P be a logic program over $\underline{\Sigma} = (\Pi, \Phi)$, that is a set $P = \{C_1, \dots, C_k\}$ of definite clauses, and let A be a goal. Then the *jungle grammar associated with P, A* is

$$\mathcal{J}(P, A) = (\mathcal{J}(P), \mathcal{J}(\emptyset, A), \underline{\Sigma}, \underline{\Phi}),$$

where $\mathcal{J}(P)$ is the set $\{\mathcal{J}(C_1), \dots, \mathcal{J}(C_k)\}$ of rewrite rules representing the clauses of P , $\mathcal{J}(\emptyset, A)$ is the fully collapsed jungle representing A , $\underline{\Sigma}$ is the signature of colors, and $\underline{\Phi} = (\{s, b\}, \Phi)$ is subsignature of terminal colors.

Theorem 4.12 states that there is a close correspondence between the refutations of a goal A in a program P and the derivations in the associated grammar.

Theorem 4.12 (Refutations as derivations and vice versa). *Let P be a logic program, A be a goal, and $\mathcal{J}(P, A)$ be the associated jungle grammar. Then there exists*

a refutation of A in P producing the computed answer substitution θ , if and only if there exists a derivation in $\mathcal{J}(P, A)$ with associated substitution σ_{tr} such that $\sigma_{tr_s} \cong \theta$.

Proof. *Only if:* By hypothesis, $A \triangleq A_0 \rightarrow_{c_1, \theta_1} A_1 \rightarrow_{c_2, \theta_2} \cdots \rightarrow_{c_n, \theta_n} A_n = \square$, and $\theta = (\theta_n \circ \cdots \circ \theta_1)|_{\text{var}(A)}$. Then by n applications of the completeness result (Theorem 4.10) we can build the rewriting $\rho = (\mathcal{J}(\emptyset, A) \triangleq G_0 \Rightarrow_{\mathcal{J}(C_1)} G_1 \Rightarrow_{\mathcal{J}(C_2)} \cdots \Rightarrow_{\mathcal{J}(C_n)} G_n)$, where $\text{FORM}(G_i) = A_i$, and $(\sigma_{tr_i})_s = \theta_i|_{\text{var}(A_{i-1})}$ for each $0 \leq i \leq n$. Since $\text{FORM}(G_n) = \square$, all the labels of G_n are in $\underline{\Phi}$; thus, ρ is a true derivation. Finally, we have $\theta = (\theta_n \circ \cdots \circ \theta_1)|_{\text{var}(A)} = \theta_n|_{\text{var}(A_{n-1})} \circ \cdots \circ \theta_1|_{\text{var}(A)} = (\sigma_{tr_n})_s \circ \cdots \circ (\sigma_{tr_1})_s = (\sigma_{tr_n} \circ \cdots \circ \sigma_{tr_1})_s = \sigma_{tr_s}$, using obvious properties of substitutions and Definition 2.7.

If: Similar to the *only if* part, exploiting the soundness. \square

Conversely, let us show that from every hyperedge replacement jungle grammar \mathcal{G} over a logic programming signature, a logic program $\mathcal{P}(\mathcal{G})$ together with a goal A can be extracted, in such a way that the one-to-one correspondence between refutations for A in $\mathcal{P}(\mathcal{G})$ and derivations of \mathcal{G} still holds.

Theorem 4.13 (From hyperedge replacement jungle grammars to logic programs). *Given a hyperedge replacement jungle grammar, $\mathcal{G} = (\mathcal{R}, G, \underline{\Sigma}, \underline{\Phi})$, where $\mathcal{R} = \{p_1, \dots, p_n\}$ and $\underline{\Sigma}$ and $\underline{\Phi}$ are as in Definition 4.11, let $\mathcal{P}(\mathcal{G})$ be the program $\{\mathcal{C}(p_1), \dots, \mathcal{C}(p_n)\}$, where $\mathcal{C}(p_i)$ is the clause associated with p_i (see Definition 4.9). Then for each derivation of \mathcal{G} with associated substitution σ_{tr} there is a refutation for $\text{FORM}(G)$ in $\mathcal{P}(\mathcal{G})$ with computed answer substitution $\theta \cong \sigma_{tr}|_{\text{FORM}(G)}$. Vice versa, for each refutation for $\text{FORM}(G)$ in $\mathcal{P}(\mathcal{G})$ with computed answer substitution θ there is a derivation of \mathcal{G} with associated substitution σ_{tr} , such that $\sigma_{tr}|_{\text{FORM}(G)} \cong \theta$.*

Proof (outline). The same soundness and completeness statements of Theorem 4.10 also hold if one starts with an arbitrary hyperedge replacement jungle rewrite rule over a logic programming signature and considers its associated definite clause, as in Definition 4.9, provided that one takes into account that some of the variables of a jungle G can disappear when considering its associated goal $\text{FORM}(G)$. Then the extension of this result to entire derivations is straightforward. \square

5. Comparison with related works

We discuss here the relationship between our representation of term rewriting and logic programming as hyperedge replacement jungle rewriting and other representations proposed in the literature. The main aspect which has to be compared with others is the handling of non-left-linear rules.

5.1. Non-left-linearity

The fact that our jungle representation of term rewrite rules enjoys completeness w.r.t. applicability also in the non-left-linear case (Proposition 3.17) clearly exploits the fact that full term unification can be expressed in our framework by pushouts. This is indeed the reason why we can model logic programming, which other representations based on the categories of graphs or hypergraphs are not able to deal with, because they can express just (a restricted form of) pattern matching. In fact, besides the problem due to non-left-linearity of rules (discussed in Section 3), another source of incompleteness arises for logic programming if one tries to represent a clause in a direct way, i.e., with the whole head of the clause in the lhs of the associated rewrite rule. For example, the clause $C \equiv (p(a) :- B_1, \dots, B_n)$ can be applied to the goal “ $p(x)$ ” (producing the substitution $\{x/a\}$), but there is no graph morphism from $\mathcal{J}(p(a))$ to $\mathcal{J}(p(x))$ (Fig. 17).

Clearly this situation is peculiar to logic programming, where the variables appearing in a goal can be instantiated during a resolution step.

It could be argued that the use of unification is too expensive for term-rewriting systems, for which pattern matching is sufficient since just the variables appearing in the rules can be instantiated. A complete analysis of the complexity of rewriting would be necessary to compare the cost of rewriting in the various approaches, but this goes beyond the scope of this paper. Nevertheless, it is worth stressing some points. In most of the other approaches, the cost of applying a rewrite rule to a term (represented as a jungle or a dag) is essentially the cost of finding an occurrence of the lhs of that rule in the term. On the contrary, in our approach one has to find an occurrence of a single function symbol (the top operator of the lhs of the rule), and then to perform a unification (the rhs pushout). In general, the unification can have far-reaching effects (as commented in Section 2.2) but the following fact, which can be checked by inspecting carefully the proof of Lemma 1.17, warrants that during the application of a left-linear rule, only “local” effects are produced. Thus, its cost is comparable to that of other approaches.

Fact 5.1 (Unification has local effects in the case of left-linear rules). *If $t \rightarrow t'$ is a left-linear rule, $\mathcal{J}(t \rightarrow t') = (L \xleftarrow{l} K \xrightarrow{r} R)$ is its jungle representation and $g: L \rightarrow G$ is a morphism, then any two items (edges or nodes) of the context jungle D (see Fig. 9) which are identified by the rhs pushout are identified with an item of t , too. In other words, the effect of the application of the rule to the term represented by G is bounded by the occurrence of t in that term.*

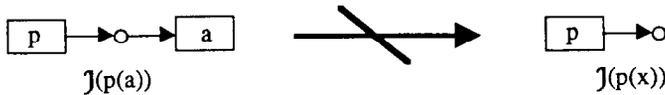


Fig. 17.

This fact does not hold for non-left-linear rules. For example, applying the rule $f(x, x) \rightarrow p(x)$ to a jungle representing the term $f(g^n(a), g^n(a))$ where the two occurrences of the subterm $g^n(a)$ are not shared, it is easy to check that the unification causes the folding of part of the graph which can be arbitrarily far from the occurrence of the lhs term (see also the example after Lemma 1.17). On the other hand, such an application would not be possible in the other approaches, which sometimes propose ad hoc mechanisms to treat non-left-linear rules.

In [2] non-left-linear rules are not allowed, while in [28] they are admitted but a much weaker result than Proposition 3.7 is presented. The papers [30, 23, 24] propose various kinds of graph rewriting in categories of graphs with partial morphism, presenting term graph rewriting as a main application. In [30] it is hinted that for non-left-linear rules the equality of subterms corresponding to distinct occurrences of the same variable in the lhs must be checked explicitly. In [23] a generalization of the notion of occurrence is proposed instead, which essentially amounts to fold the graph to be rewritten at least as much as it is necessary to find a (classical) occurrence of the lhs of the rule. In [24], Kennaway does not discuss non-left-linear rules, but it proposes to perform rewriting as a single pushout in the category of jungles with partial morphisms, and discusses interesting properties of this category: results similar to ours should be easily restated in his framework. Both [2] and [23] stress that non-left-linearity is much more important in logic programming than in TRSs.

As emphasized in various places, the works of Hoffman, Plump and others [29, 20–22] have been a major source of inspiration for ours. Since we use their notion of jungles, the representation of terms is identical. The main difference is that they perform the double pushout construction in category \mathbf{HGraph}_{Σ} , while we do it in category \mathbf{Jungle}_{Σ} .

In order to deal with non-left-linear rules, they propose to add to the graph-rewriting system representing a TRS a collection of *folding rules*, one for each operator of the signature, which have the effect of merging together subjungles which represent identical subterms. In the case of Example 3.8, the two distinct edges labeled by “0” of G would be merged by a folding rule, and the resulting jungle G' could be rewritten by the standard representation of rule $rem(x, cons(x, y)) \rightarrow rem(x, y)$. Folding rules have also the effect of improving the efficiency of term graph rewriting (thanks to a result analogous to Theorem 3.6), because they increase the degree of subterm sharing.

For what concerns the cost of the application of non-left-linear rules, it is not possible here to compare their approach to ours in depth. We just observe that the overall cost of the application of a non-left-linear rule in [21, 22] includes the cost of folding and, thus, should be comparable with ours. However, also folding which can result a posteriori unnecessary must be performed in their approach, in order to warrant the applicability of all rewrite rules.

5.2. Logic programming as graph rewriting

A related work is reported in [8, 9], where context-free hypergraph grammars are used for analyzing recursive definitions, and are applied as a case study to a proper subset of logic programs, i.e., recursive queries in relational databases.

Two papers by the authors in joint work with others [5, 7] propose a representation of logic programming via graph rewriting. The representation introduced in [5] follows essentially the same guidelines as that presented in Section 4, but formulas were represented by a hybrid of dags (for the functional part) and jungles (for the predicate symbols). This caused some awkwardness in the presentation. However, this problem has been solved in the cleaner formulation contained here, which has also the relevant advantage of being an application of a more general framework.

The representation proposed in [7], on the other hand, is quite different, since a resolution step is modeled by three pushouts instead of two, and clauses are represented by context-dependent rewrite rules. The advantage of this second representation is that the unification and the rewriting phase of a resolution step are kept separated, and that results of the theory of graph grammars can be applied easily to prove the correctness of some operations on clauses, like unfolding and partial evaluation. Such results have not been explored yet for the representation of logic programs proposed in Section 4.

6. Conclusions and future work

In this paper we introduced hyperedge replacement jungle rewriting, and investigated its expressive power by showing that it can model both term-rewriting systems and logic programming in a faithful way. For term-rewriting systems we proved the soundness of their jungle representation, and a result of completeness w.r.t. applicability which is stronger than similar results in the related literature, since it works also for non-left-linear rules. For logic programming both soundness and completeness hold.

The uniform treatment of term-rewriting systems and logic programming in the same framework allows one to analyze similarities and differences of the two formalisms, and suggests the possibility of merging them in a clean way. For example, it should be possible to represent logic programming modulo an equational theory by putting together the jungle rules representing the logic program and the rules which represent a term-rewriting system for the equational theory.

We also showed that (under different hypotheses) from a hyperedge replacement jungle-rewriting system one can extract either a TRS or a logic program. This suggests that the expressive power of hyperedge replacement jungle rewriting should not be much greater than a suitable combination of the two formalisms. It could correspond to some kind of term rewriting with unification.

In this paper we considered acyclic jungles only. A generalization of our formalism could permit cyclic jungles as well. This would allow us to model the rewriting of

rational terms and, on the logic programming side, to represent unification without occur-check, as in Prolog II [4].

The comparison with the single-pushout approach [30, 23, 24, 27] is another direction for future investigation. For example, Kennaway [24] shows that the single-pushout approach in the category of jungles with partial morphisms strictly subsumes the double-pushout approach in the same category with total morphisms.

Finally, the theory of graph-grammars provides a rich collection of results about the parallel and concurrent combination of rewrite rules [10, 25]. The formulation of similar results for the class of hyperedge replacement jungle-rewriting systems should be explored (some related results has been presented in [7]).

Acknowledgment

We thank Ugo Montanari, Michael Löwe, Hartmut Ehrig, and Francesco Parisi-Presicce for many fruitful discussions about the relationship between graph rewriting and logic programming.

References

- [1] A. Asperti and S. Martini, Projections instead of variables, a category theoretic interpretation of logic programs, in: *Proc. 6th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1989), 337–352.
- [2] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer and M.R. Sleep, Term graph reduction, in: *Proc. PARLE*, Lecture Notes in Computer Science, Vol. 259 (Springer, Berlin, 1987) 141–158.
- [3] M. Bauderon and B. Courcelle, Graph expressions and graph rewritings, *Math. Systems Theory* **20** (1987) 83–127.
- [4] A. Colmerauer, PROLOG II – Reference Manual and Theoretical Model, Internal Report, Groupe Intelligence Artificielle, Université Aix-Marseille II, 1982.
- [5] A. Corradini, U. Montanari, F. Rossi, H. Ehrig and M. Löwe, Logic programming and graph grammars, in: [13], 221–237.
- [6] A. Corradini and F. Rossi, Hyperedge replacement jungle rewriting and its relationship to term rewriting systems, in: *Proc. SEMAGRAPH Symp.*, Nijmegen, Lecture Notes in Computer Science (Springer, Berlin, 1992), to appear.
- [7] A. Corradini, F. Rossi and F. Parisi-Presicce, Logic programming as hypergraph rewriting, in: *Proc. CAAP '91*, Lecture Notes in Computer Science, Vol. 493 (Springer, Berlin, 1991) 275–295.
- [8] B. Courcelle, On using context-free graph grammars for analyzing recursive definitions, in: K. Fuchi and L. Kott, eds., *Programming of Future Generation Computers II* (Elsevier, Amsterdam, 1988) 83–122.
- [9] B. Courcelle, Recursive queries and context-free graph grammars, *Theoret. Comput. Sci.* **78** (1988) 217–244.
- [10] H. Ehrig, Aspects of concurrency in graph grammars, in [14], 58–81.
- [11] H. Ehrig, Tutorial introduction to the algebraic approach of graph-grammars, in [15], 3–14.
- [12] H. Ehrig, A. Habel, H.-J. Kreowski and F. Parisi-Presicce, From graph grammars to high-level replacement systems, in [13], 269–291.
- [13] H. Ehrig, H.-J. Kreowski and G. Rozenberg, eds., *Proc. 4th Internat. Workshop on Graph-Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 532 (Springer, Berlin, 1991).

- [14] H. Ehrig, M. Nagl and G. Rozenberg, eds., *Proc. 2nd Internat. Workshop on Graph-Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 153 (Springer, Berlin, 1983).
- [15] H. Ehrig, M. Nagl, G. Rozenberg and A. Rosenfeld, eds., *Proc. 3rd Internat. Workshop on Graph-Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987).
- [16] H. Ehrig, M. Pfender and H.J. Schneider, Graph-grammars: an algebraic approach, in: *Proc. IEEE Conf. on Automata and Switching Theory* (1973) 167–180.
- [17] J.A. Goguen, What is unification? A categorical view of substitution, equation and solution, in: M. Nivat and H. Ait-Kaci, eds., *Resolution of Equations in Algebraic Structures* (Academic Press, New York, 1989).
- [18] A. Habel, Hyperedge replacement: grammars and languages, Ph.D. Thesis, University of Bremen, 1989.
- [19] A. Habel and H.-J. Kreowski, May we introduce to you: hyperedge replacement, in [15], 15–26.
- [20] A. Habel, H.-J. Kreowski and D. Plump, Jungle evaluation, in: *Proc. 5th Workshop on Specification of Abstract Data Types*, Lecture Notes in Computer Science, Vol. 332 (Springer, Berlin, 1988) 92–112.
- [21] B. Hoffmann and D. Plump, Jungle evaluation for efficient term rewriting, in: *Proc. Algebraic and Logic Programming*, Lecture Notes in Computer Science, Vol. 343 (Springer, Berlin, 1988) 191–203.
- [22] B. Hoffmann and D. Plump, Implementing term rewriting by jungle evaluation, *RAIRO Théor. Inform.*, to appear.
- [23] J.R. Kennaway, On “on graph rewritings”, *Theoret. Comput. Sci.* **52** (1987) 37–58.
- [24] J.R. Kennaway, Graph rewriting in some categories of partial morphisms, in [13], 490–504.
- [25] H.-J. Kreowski, Is parallelism already concurrency? Part 1: derivations in graph grammars, in [15], 343–360.
- [26] J.W. Lloyd, *Foundations of Logic Programming* (Springer, Berlin, 1984); 2nd ed., 1987.
- [27] M. Löwe, Extended algebraic graph transformation, Ph.D. Thesis, Fachbereich Informatik, Technische Universität Berlin, 1991.
- [28] F. Parisi-Presicce, H. Ehrig and U. Montanari, Graph rewriting with unification and composition, in [15], 496–514.
- [29] D. Plump, Im Dschungel: Ein neuer Graph-Grammatik-Ansatz zur effizienten Auswertung rekursiv definierter Funktionen, Diplomarbeit, Fachbereich Mathematik–Informatik, Universität Bremen, 1986.
- [30] J.C. Raoult, On graph rewritings, *Theoret. Comput. Sci.* **32** (1984) 1–24.
- [31] D.E. Rydeheard and R.M. Burstall, A categorical unification algorithm, in: *Proc. Workshop on Category Theory and Computer Programming*, Lecture Notes in Computer Science, Vol. 240 (Springer, Berlin, 1985).