
Proceedings Paper:
Habel, Annegret, Kreowski, Hans-Jörg and Plump, DETLEF (1988) Jungle Evaluation. In: Recent Trends in Data Type Specification (WADT 1987), Selected Papers. Lecture Notes in Computer Science. Springer, pp. 92-112.

https://doi.org/10.1007/3-540-50325-0_5

JUNGLE EVALUATION

Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump*
Fachbereich Mathematik und Informatik
Universität Bremen
D–2800 Bremen 33

Abstract

Jungle evaluation is proposed as a new graph rewriting approach to the evaluation of functional expressions and, in particular, of algebraically specified operations. Jungles – being intuitively forests of coalesced trees with shared substructures – are certain acyclic hypergraphs (or equivalently, bipartite graphs) the nodes and edges of which are labeled with the sorts and operation symbols of a signature. Jungles are manipulated and evaluated by the application of jungle rewrite rules, which generalize equations or, more exactly, term rewrite rules. Indeed, jungle evaluation turns out to be a compromise between term rewriting and graph rewriting displaying some favorable properties: the inefficiency of term rewriting is partly avoided while the possibility of structural induction is maintained, and a good part of the existing graph grammar theory is applicable so that there is some hope that the rich theory of term rewriting is not lost forever without a substitute.

1 Introduction

Whenever an algebraic specification is meant to be a design specification and as such to solve a data-processing problem, it must be executable. The demand of executability distinguishes a solution from the problem, a design specification from the requirement definition. Hence, the issue of operational semantics has been of great interest for more than ten years within the algebraic approach to software development. Algebraic specifications with operational semantics are meaningful in the following respects:

(1) Interpreters can be built as tools supporting the process of software development.

(2) Tests can be run at a comparatively early stage of software development.

(3) Algebraic specifications can be used as prototypes of software systems.

(4) Algebraic specifications can be considered as programs in an algebraic programming language.

(5) Algebraic specifications may be employed as rule–based systems within expert systems.

The favorite approach to equip algebraic, particularly equational specifications with operational semantics is term rewriting. This is not surprising because equations as the key concept of the algebraic–specification technique consist of terms and may be seen as left–to–right term rewrite rules. Moreover, the theoretical foundations of and practical experience with term rewriting are tremendous and a lot of knowledge is available how termination, confluence and related concepts work.

Nevertheless, there are reasons for concern: among them we would like to point at the possible inefficiency of term rewriting. The execution of algebraic specifications may be extremely slow compared with the evaluation of functional or logical programs. If the solved problems are hard, nothing can be done. But what if the framework must be blamed?

*Work of this author is partially supported by the ESPRIT–project PROSPECTRA, ref. # 390.

Consider, for example, the specification of a function generating a totally balanced binary tree (without labels) of height n if n is the input.

$$\begin{array}{ll} \textbf{generate} = \textbf{nat} + \textbf{bintree}+ & \\ \text{opns}: & GEN : nat \to bintree \\ \text{eqns}: & GEN(0) = BIN(EMPTY, EMPTY) \\ & GEN(SUCC(N)) = BIN(GEN(N), GEN(N)) \end{array}$$

Concerning the imported specifications we assume that **nat** provides a sort *nat*, a constant(symbol) 0 and a unary operation(symbol) *SUCC* at least and that **bintree** provides a sort *bintree*, a constant(symbol) *EMPTY* and a binary operation(symbol) *BIN* at least. Now, the evaluation of the term $GEN(SUCC^n(0))$ for some $n \geq 0$ within the framework of term rewriting (or, likewise, of tree rewriting) consumes time and space or a number of processors exponential in n.

This unfortunate behavior is easily avoided as long as the two identical subterms of the right–hand side of the second equation do not cause a duplication of computation.

The recipe is clear: Represent functional expressions by structures with shared substructures rather than by terms or trees. This is the basic idea of graph grammar approaches to the evaluation of functional expressions as studied by Ehrig, Padawitz, Rosen, Staples and – most recently – by Barendregt, van Eekelen, Glauert, Kennaway, Plasmeijer, and Sleep (see [ER 76], [St 80], [Pa 82], [BEGKPS 87]). Unfortunately, these approaches have some drawbacks. The types of graphs and graph rewrite rules introduced and used are exotic in the sense that the major body of graph grammar theory is not applicable. Moreover, there is no structural induction available which has been proved so extremely helpful in term and tree rewriting.

Therefore, we feel encouraged to introduce a new and alternative graph grammar approach to the evaluation of functional expressions and, in particular, of algebraically specified operations. We propose jungle evaluation the characteristics of which are the following:

(1) If the reader understands the intuition behind the notions of a *tree* and a *forest* in the sense of graph theory or computer science, he or she may think of a *jungle* as a forest of "coalesced trees" with shared substructures.

(2) Formally, jungles are recursively generated acyclic (hyper-)graphs so that structural induction is available.

(3) Jungle evaluation is intentionally related to the evaluation of algebraic specifications.

(4) Jungle evaluation can be seen as graph rewriting in the sense of the "Berlin–approach" so that various known results on graph grammar derivations can be applied (see, e.g., [Eh 79] and [Kr 87]).

(5) Especially, jungle evaluation comprises modes of non–sequential rewriting.

The paper is organized in the following way. In section 2, jungles are defined, and some of their basic properties are established while their relationship to terms is considered in section 3. Jungle rewriting is the topic of section 4 including the main result of the paper which presents sufficient conditions on (hyper-)graph rewriting rules so that their application to jungles yields jungles again. In section 5, some first evidence is presented that jungle evaluation can simulate term evaluation in a meaningful and efficient way. Section 6 contains a short concluding discussion. Finally, the basic notions on (hyper-)graphs and (hyper-)graph rewriting are recalled in the appendix as far as they are needed in this paper.

2 Jungles

In this section we define jungles as special hypergraphs that reflect the typing of a given signature. Intuitively, jungles are forests with interwoven trees. This structure ensures that each node in a jungle represents a unique term (cf. section 3). Moreover, it is possible to characterize jungles by a set of generating rules. As a consequence, a structural induction principle and efficient syntax analysis for jungles are available.

2.1 General Assumption

In the following we consider hypergraphs over an arbitrary, but fixed signature $SIG = (S, OP)$, i.e., nodes are labeled by sorts from S and hyperedges are labeled by operation symbols from OP. □

2.2 Definition (Jungle)

A hypergraph G is a *jungle (over SIG)* if

1. G is acyclic,[1]
2. $outdegree_G(v) \leq 1$ for each $v \in V_G$,[2]
3. the labeling of G is *compatible* with SIG, i.e., for each $e \in E_G$, $m_G(e) = op : s_1 \dots s_n \rightarrow s$ implies $l^*_G(s_G(e)) = s$ and $l^*_G(t_G(e)) = s_1 \dots s_n$.

Remarks

1. Trees and forests over SIG are special jungles where the indegree of each node is at most one.

2. For each hyperedge in a jungle labeled by an operation symbol $op : s_1 \dots s_n \rightarrow s$ there is a unique source node labeled by s and a sequence of n (not necessarily distinct) target nodes labeled by $s_1, \dots, s_n$. The sequence of target nodes is empty if op is a constant symbol.

3. Each subhypergraph of a jungle is a jungle, too. Such a subhypergraph is called a *subjungle*. □

2.3 Example

The following hypergraph is a jungle over the signature of the specification **generate** (where "bin" stands for "*bintree*").

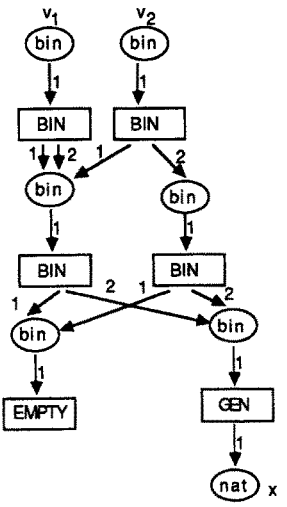


Figure 2.1: A Jungle

□

[1] A hypergraph G is said to be acyclic if the underlying bipartite graph $U(G)$ is acyclic.

[2] For a hypergraph G and a node $v \in V_G$, $indegree_G(v)$ denotes the number of "tentacles" incoming in v and $outdegree_G(v)$ denotes the number of "tentacles" outgoing from v, i.e., $indegree_G(v) = \sum_{e \in E_G} \#(v, t_G(e))$ and $outdegree_G(v) = \sum_{e \in E_G} \#(v, s_G(e))$, where $\#(a, w)$ denotes the number of occurrences of an element a in a sequence w.

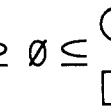
Beside the graph-theoretic description given in Definition 2.2 jungles can be characterized by three kinds of jungle generating rules. Theorem 2.5 shows that each jungle is generated from the empty hypergraph $\emptyset$ (which is a jungle) by repeated application of these rules. (Note that general hypergraph rules and their application are discussed in the appendix.)

2.4 Definition (Jungle Generating Rules)
The set *GEN* of *jungle generating rules* consists of the following rules:

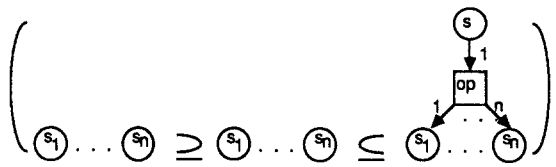
(Variable Generation)[3]

$< s > =$  for each sort $s \in S$,

(Constant Generation)

$< c : \rightarrow s > =$  for each constant symbol $c : \rightarrow s$,

(Operation Application)

$< op : s_1 \ldots s_n \rightarrow s > =$  for each operation symbol $op : s_1 \ldots s_n \rightarrow s$.

□

Figure 2.2 (see next page) shows how the jungle of Figure 2.1 can be generated. It turns out that the set of all hypergraphs derivable from the empty hypergraph by *GEN* and the set of all jungles are equal.

2.5 Theorem (Characterization of Jungles)
A hypergraph G is a jungle if and only if $\emptyset \overset{*}{\underset{GEN}{\Longrightarrow}} G$.

Proof
It is simple to check that each of the rules, when applied to a jungle, preserves the conditions of Definition 2.2. Hence each derivation with these rules starting from the empty jungle yields a jungle.

Now let G be a jungle. We show $\emptyset \overset{*}{\underset{GEN}{\Longrightarrow}} G$ by induction on the number n of nodes in G. Obviously the proposition holds for $n = 0$ since then G is the empty jungle. Let therefore $n \geq 1$ and assume $\emptyset \overset{*}{\underset{GEN}{\Longrightarrow}} \overline{G}$ for each jungle $\overline{G}$ with less than n nodes. Because G is non-empty and acyclic, there is a node v in G with $indegree_G(v) = 0$. Let $\overline{G}$ be the subjungle of G induced by $V_G - \{v\}$. The following case analysis shows that $\overline{G} \underset{GEN}{\Longrightarrow} G$ which by the induction hypothesis implies $\emptyset \overset{*}{\underset{GEN}{\Longrightarrow}} G$.

Case 1: $outdegree_G(v) = 0$. Then $\overline{G} \underset{<s>}{\Longrightarrow} G$, where s is the label of v.

Case 2: $outdegree_G(v) = 1$. Let e be the unique hyperedge in G with $s_G(e) = v$. By construction of $\overline{G}$ we have $E_{\overline{G}} = E_G - \{e\}$.

Case 2.1: $m_G(e)$ is a constant symbol $c :\rightarrow s$. Then $l^*_G(s_G(e)) = s$ and $t_G(e) = \lambda$ imply $\overline{G} \underset{<c:\rightarrow s>}{\Longrightarrow} G$.

Case 2.2: $m_G(e)$ is an operation symbol $op : s_1 \ldots s_n \rightarrow s$. Then all nodes in $t_G(e)$ belong to $\overline{G}$. Consequently $l^*_G(t_G(e)) = s_1 \ldots s_n$ implies that the target nodes of e constitute an occurrence of the left-hand side of $< op : s_1 \ldots s_n \rightarrow s >$ in $\overline{G}$. Further $l^*_G(s_G(e)) = s$ holds, so $\overline{G} \underset{<op:s_1 \ldots s_n \rightarrow s>}{\Longrightarrow} G$. □

[3]The notation refers to the relation between jungles and terms with variables which is discussed in section 3.

2.6 Example

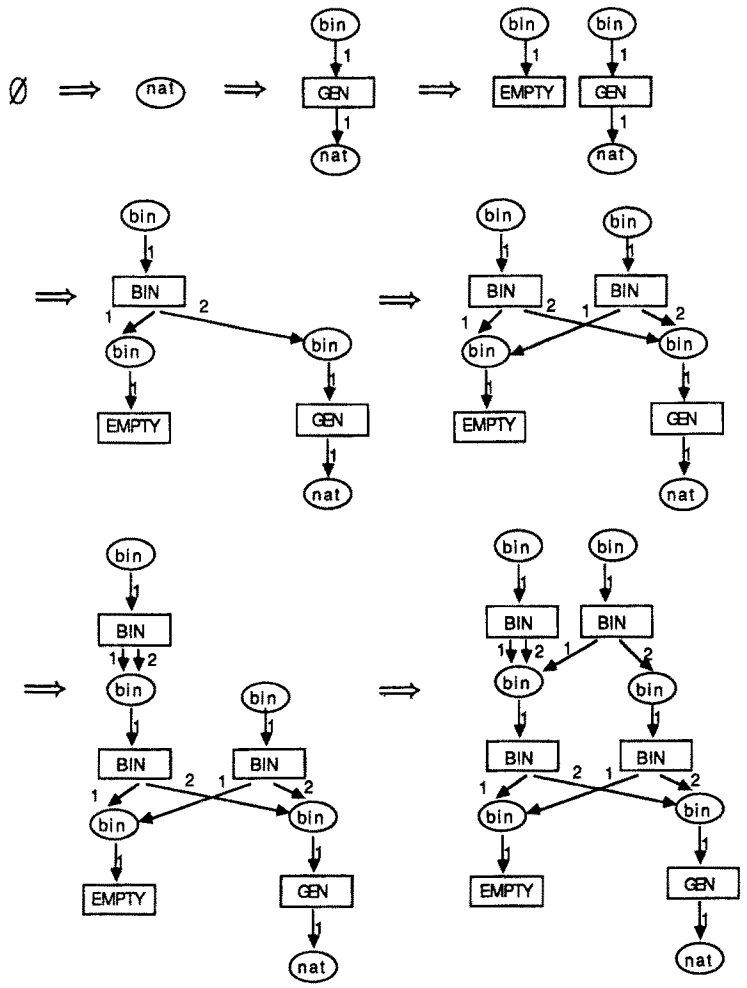


Figure 2.2: Generation of a Jungle

□

Theorem 2.5 establishes a structural induction principle.

2.7 Corollary (Structural Induction)
A property P holds for all jungles provided that

1. P holds for the empty jungle, and
2. for all jungles G and H, if P holds for G and $G \underset{GEN}{\Longrightarrow} H$, then P holds for H also.

Proof
By Theorem 2.5 for all non-empty jungles G there are derivation sequences of the form $\emptyset \underset{GEN}{\Longrightarrow} G_1 \underset{GEN}{\Longrightarrow} \ldots \underset{GEN}{\Longrightarrow} G_n = G$. Induction on the lengths of these sequences yields the desired result. □

For testing whether or not a given hypergraph is a jungle one may check acyclicity, the outdegree condition, and the labeling condition given in Definition 2.2. By Theorem 2.5 there is a more efficient method based on the set GEN^{-1} of all inverse rules of jungle generating rules.

2.8 Corollary (Syntax Analysis)
If SIG is finite,[4] the algorithm

apply rules from GEN^{-1} as long as possible

reduces a given hypergraph G to the empty hypergraph if and only if G is a jungle.

The algorithm consumes linear time provided that hypergraphs are represented in such a way that the applicability of each rule in GEN^{-1} can be tested in constant time.

Proof
If SIG is finite, GEN^{-1} contains a fixed number of rules. Since the applicability of each rule can be checked in constant time, the test whether any of the rules is applicable to a given hypergraph can be performed in constant time, too. Furthermore, each application of a rule removes a node, so each derivation stops after at most n steps, where n is the number of nodes in the given hypergraph. Hence the above algorithm terminates in linear time.

To show that the algorithm is correct, let G, H be hypergraphs and $G \overset{*}{\underset{GEN^{-1}}{\Longrightarrow}} H$ be a derivation such that no rule in GEN^{-1} can be applied to H. Then $H \overset{*}{\underset{GEN}{\Longrightarrow}} G$ implies that G is a jungle in case $H = \emptyset$. If H is non-empty, there is no derivation of the form $H \overset{*}{\underset{GEN^{-1}}{\Longrightarrow}} \emptyset$ and consequently no derivation $\emptyset \overset{*}{\underset{GEN}{\Longrightarrow}} H$. Thus H is not a jungle by Theorem 2.5. □

3 Relating Terms and Jungles

The jungles introduced and investigated in the previous section are closely related to terms. On one hand, each node of a jungle represents uniquely a term. On the other hand, there are several possibilities for representing a term as a jungle. A natural and simple representation is by a "variable–collapsed" tree. Another distinguished, useful representation (exploiting the possibility of structure sharing exhaustively) is by a "fully collapsed" tree. Beside these presentations of terms there are lots of other jungles representing a given term. When translating term rewrite rules into jungle rules we make use of the different term presentations (compare Section 5).

Jungles over a signature SIG are special hypergraphs the nodes and hyperedges of which are labeled by sorts and operation symbols, respectively. Fixing a node in a jungle, a term may be extracted from the jungle by (a) taking the label of the only outgoing hyperedge and gathering the terms of the target nodes (of the hyperedge) and (b) taking the node as a variable provided that there is no outgoing hyperedge. In this way, each node of a jungle represents a term. Given a jungle G, the nodes without outgoing hyperedges play a specific role: with respect to terms they represent variables. To indicate this role, they are called *variable nodes*. The set of these nodes is denoted by VAR_G.

3.1 Theorem (Terms Represented by a Jungle)
Let G be a jungle. Then for each node $v \in V_G$ the following construction yields a unique term from $T_{SIG}(VAR_G)$, denoted by $term_G(v)$:

- $term_G(v) = v$ if $v \in VAR_G$, i.e., there is no $e \in E_G$ with $s_G(e) = v$,
- $term_G(v) = c$ if there is $e \in E_G$ with $s_G(e) = v$, $t_G(e) = \lambda$, and $m_G(e) = c$,
- $term_G(v) = op(term_G(v_1), \ldots, term_G(v_n))$ if there is $e \in E_G$ with $s_G(e) = v$, $t_G(e) = v_1 \ldots v_n$, and $m_G(e) = op$.

[4] $SIG = (S, OP)$ is said to be finite if both S and OP are finite sets.

Moreover, $l_G(v)$ is the sort of $term_G(v)$.

Remarks

1. By jungle properties, the construction yields a *unique* term: The outdegree property (the outdegree of each node in a jungle is at most one) as well as the cycle-freeness property (each jungle is acyclic) are essential.

2. By Theorem 3.1 each node of a jungle refers to a term which may contain variables. In particular, each node without outgoing hyperedges represents a variable. Thus, these nodes are called variable nodes. □

3.2 Example

Considering the jungle given in Figure 2.1, the nodes v_1 and v_2 both represent the term $BIN(BIN(EMPTY, GEN(x)), BIN(EMPTY, GEN(x)))$. □

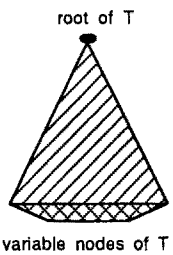
According to Theorem 3.1 , each jungle represents a collection of terms; each node of a jungle represents *one* term. Vice versa, we want to represent terms by jungles. For this purpose we consider special jungles, called *variable-collapsed trees* and *fully collapsed trees.* These jungles possess exactly one node without incoming hyperedges which is called *root.* It turns out that each term can be represented by a variable-collapsed tree and by a fully collapsed tree the root of which represents the term.

3.3 Definition (Variable–Collapsed Tree)

A jungle T is called a *variable-collapsed tree* if there is exactly one node in T, denoted by $root_T$, with $indegree_T(root_T) = 0$ and $indegree_T(v) = 1$ for each $v \in V_T - (VAR_T \cup \{root_T\})$.

Remark

In variable–collapsed trees, non-variable nodes are required to satisfy a restrictive indegree condition (like the indegree condition for trees) while variable nodes are not required to satisfy this condition. In this sense, a variable–collapsed tree may be seen as a tree in which some of the leaves are collapsed. This is depicted as follows:



□

3.4 Theorem (Representation of Terms)

For each term t, there is a (up to isomorphism) unique variable–collapsed tree T satisfying $term_T(root_T) = t$.

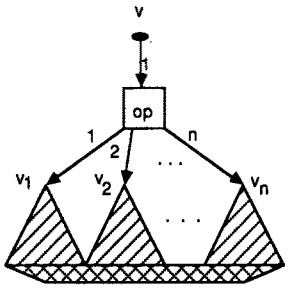
Sketch of Proof

Given a term t with n operation symbols, a variable–collapsed tree T satisfying $term_T(root_T) = t$ may be constructed according to the following rules:

(1) For each variable of sort s, a variable node with label s is generated. Let $jungle_0(t)$ be the obtained jungle. Then it represents the set of all variables of t.

(2) Suppose $jungle_k(t)$ is already constructed representing the set of all variables as well as the multi-set of all occurrences of subterms with at most k (but at least one) operation symbols. If $k = n$, choose T as $jungle_k(t)$. Otherwise we construct a jungle, denoted by $jungle_{k+1}(t)$, which represents the set of all variables and the multi-set of all subterms with at most $k + 1$ (but at least one) operation symbols. This may be done as follows: Let t_0 be a subterm of t with $k + 1$ operation symbols. Then t_0 is of the form

$op(t_1,\ldots,t_n)$ where $t_1,\ldots,t_n$ possess at most k operation symbols. Since $t_1,\ldots,t_n$ are subterms of t, there are nodes $v_1,\ldots,v_n$ in $jungle_k(t)$ representing $t_1,\ldots,t_n$, respectively. Suitable application of the jungle generating rule $< op : s_1 \ldots s_n \rightarrow s >$ to $jungle_k(t)$ yields a jungle with a node v representing the term $t_0 = op(t_1,\ldots,t_n)$.



Analogously, all (occurrences of) subterms with $k+1$ operation symbols may be handled.

Notation
The variable–collapsed tree T corresponding to t is denoted by $jungle(t)$. □

3.5 Definition (Fully Collapsed Jungle and Fully Collapsed Tree)
1. A jungle T is said to be *fully collapsed* if for all $v, v' \in V_T$, $term_T(v) = term_T(v')$ implies $v = v'$.

2. A fully collapsed jungle T is called a *fully collapsed tree* if there is exactly one node in T, denoted by $root_T$, with $indegree_T(root_T) = 0$. □

3.6 Theorem (Representation of Terms)
For each term t there is a (up to isomorphism) unique fully collapsed tree T with $term_T(root_T) = t$.

Sketch of Proof
Let $SUB(t)$ be the set of all subterms of t. Choose $V_T = SUB(t)$ and $E_T = SUB(t) - VAR(t)$.[5] For each $e = op(t_1,\ldots,t_n) \in E_T$ define $s_T(e) = e$, $t_T(e) = t_1 \ldots t_n$, and $m_T(e) = op$. Further let l_T map each subterm of t to its sort. Then $T = (V_T, E_T, s_T, t_T, l_T, m_T)$ is a fully collapsed tree with $root_T = t$ and $term_T(root_T) = t$. Moreover, for any fully collapsed tree T' with $term_{T'}(root_{T'}) = t$ an isomorphism $g : T' \rightarrow T$ is defined for all $v \in V_{T'}$ and all $e \in E_{T'}$ by $g_V(v) = term_{T'}(v)$ and $g_E(e) = term_{T'}(s_{T'}(e))$.

Notation
The fully collapsed tree T corresponding to t is denoted by $JUNGLE(t)$. □

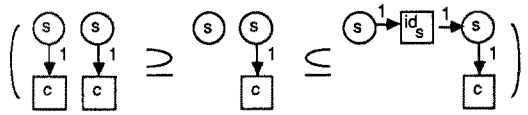
Beside the variable–collapsed tree $jungle(t)$ and the fully collapsed tree $JUNGLE(t)$ there are several other jungles (with one root) which represent the term t. These may be obtained from $jungle(t)$ by "folding" and from $JUNGLE(t)$ by "unfolding". Among all jungles representing t, $JUNGLE(t)$ is the most efficient representation of t: equal subterms are represented only once. Jungles may be folded by the application of rules called *folding* rules. These rules make use of auxiliary operation symbols indicating the equality of terms.

3.7 Definition (Folding and Unfolding Rules)
Let $SIG = (S, OP)$ be extended to the signature $SIG' = (S, OP')$ where OP' contains OP and an operation symbol $id_s : s \rightarrow s$ for each sort $s \in S$.

1. The set $FOLD$ of *folding rules* consists of the following rules:

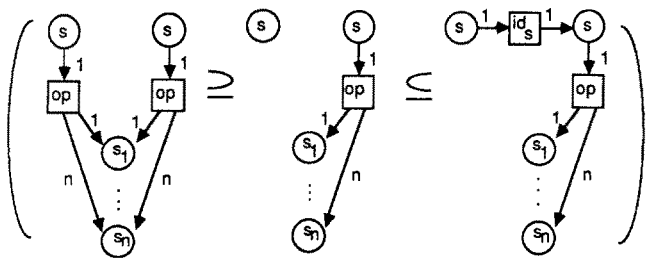
[5] $VAR(t)$ denotes the set of all variables in t.

(Constant Identification)



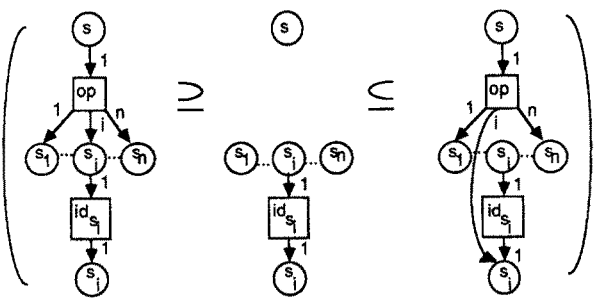
for each constant symbol $c : \rightarrow s$,

(Operation Identification)



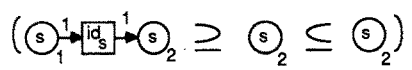
for each operation symbol
$op : s_1 \ldots s_n \rightarrow s$,

(Id–Bridging)



for each operation symbol
$op : s_1 \ldots s_n \rightarrow s$ and each $i \in \{1, \ldots, n\}$,

(Id–Elimination)



for each sort $s \in S$.

2. The set *UNFOLD* of *unfolding rules* consists of the inverse rules of *FOLD*.

Remark
Folding rules as well as unfolding rules are jungle rules in the sense of Definition 4.2. By the Jungle Preservation Theorem given in 4.3, the application of these rules to jungles yields jungles again. □

3.8 Theorem (Jungle Variation)
Let $\mathcal{J}_{SIG}$ denote the set of all jungles with one root over the signature *SIG*. Let t be a term over *SIG* and $\mathcal{J}(t)$ be the set of all jungles in $\mathcal{J}_{SIG}$ representing t. Then

$$\begin{aligned}\mathcal{J}(t) &= \{J \in \mathcal{J}_{SIG} \mid jungle(t) \overset{*}{\underset{FOLD}{\Longrightarrow}} J\} \\ &= \{J \in \mathcal{J}_{SIG} \mid J \overset{*}{\underset{FOLD}{\Longrightarrow}} JUNGLE(t)\} \\ &= \{J \in \mathcal{J}_{SIG} \mid JUNGLE(t) \overset{*}{\underset{UNFOLD}{\Longrightarrow}} J\}.\end{aligned}$$

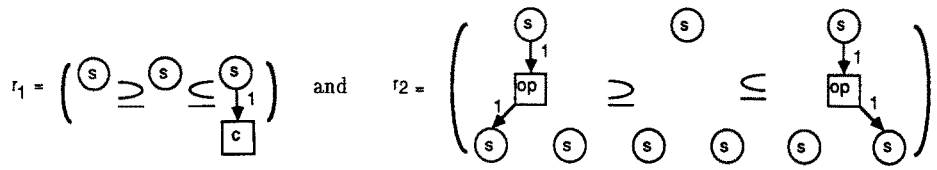
Sketch of Proof
Let $|G|$ denote the number of nodes and edges in a jungle G. Then we have $|jungle(t)| \geq |J| \geq |JUNGLE(t)|$ for each $J \in \mathcal{J}(t)$. Moreover, $jungle(t)$ and $JUNGLE(t)$ are uniquely determined, i.e., for all J_{max} and all J_{min} in $\mathcal{J}(t)$, $|J_{max}| = |jungle(t)|$ and $|J_{min}| = |JUNGLE(t)|$ implies $J_{max} \cong jungle(t)$ and $J_{min} \cong JUNGLE(t)$. Now, given $J \in \mathcal{J}(t)$ with $|jungle(t)| > |J|$ one can show that there is $J' \in \mathcal{J}(t)$ such that $|J'| > |J|$ and $J' \overset{*}{\underset{FOLD}{\Longrightarrow}} J$. This proves that each jungle in $\mathcal{J}(t)$ can be obtained from $jungle(t)$ by folding. Furthermore, for each $J \in \mathcal{J}(t)$ with $|J| > |JUNGLE(t)|$ there is $J' \in \mathcal{J}(t)$ such that $|J| > |J'|$ and $J \overset{*}{\underset{FOLD}{\Longrightarrow}} J'$. So each jungle in $\mathcal{J}(t)$ can be reduced to $JUNGLE(t)$ by folding. The rest follows from the fact that the *UNFOLD* rules are the inverses of the *FOLD* rules. □

4 Jungle Preservation

From an implementation point of view, a jungle can be regarded as a record structure (where a hyperedge corresponds to a list of pointers). Therefore, beside function evaluation, tasks like garbage collection or handling of indirect pointers should be implementable by transformations on jungles. To provide rule–based descriptions of such transformations it is desirable to have a wide spectrum of jungle manipulating rules. However, in 4.1 it is shown that the class of all jungles is not closed under applications of arbitrary hypergraph rules the components of which are jungles. In Definition 4.2 we require two weak conditions for such rules to be *jungle rules*. Theorem 4.3 states that derivations through jungle rules preserve jungles.

4.1 Preliminary Considerations

Consider the hypergraph rules



the components of which are jungles. Unfortunately, we cannot be sure that the application of these rules to a jungle yields a jungle again. Application of rule r_1 to the jungle given in Figure 4.1(1) yields a hypergraph in which the outdegree condition required for jungles is violated. Application of rule r_2 to the jungle in Figure 4.1(2) yields a hypergraph in which the cycle–freeness condition is violated.

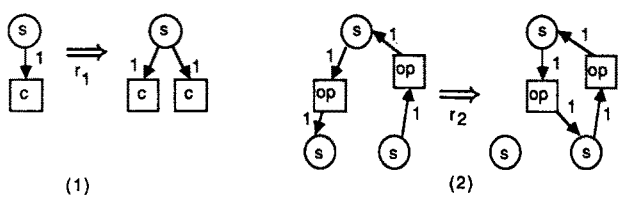


Figure 4.1: Derivations from Jungles to Hypergraphs which are not Jungles

What are the reasons?

(1) Rule r_1 appends a hyperedge to a variable node. Applying this rule to a node with an outgoing hyperedge, the outdegree condition is violated. Hence we have to forbid this kind of rule.

(2) The right–hand side of rule r_2 contains a path[6] from a gluing node v to a gluing node x without outgoing hyperedges while the left–hand side does not possess a path between these nodes. By applying

[6] Let v_1, v_2 be two nodes in a hypergraph G. Then there is a *path* from v_1 to v_2 in G if there is a path from v_1 to v_2 in the underlying bipartite graph $U(G)$.

the rule to a jungle containing a path from the image of x to the image of v, one gets a cycle. Requiring that whenever there is a path from a gluing node v to a gluing node x without outgoing hyperedges in the right–hand side there must be a path from v to x already in the left–hand side, the application of the rule cannot generate a cycle in the given jungle. □

4.2 Definition (Jungle Rule)
A rule $r = (L \supseteq K \subseteq R)$ with jungles L, K, and R is called a *jungle rule* if the following conditions are satisfied:

1. For each $x \in V_K$, $x \in VAR_L$ implies $x \in VAR_R$.
2. For each $v \in V_K$ and each $x \in V_K \cap VAR_L$, $v >_R x$ implies $v >_L x$.[7]

Examples and Remarks

1. All jungle generating rules shown in 2.4 and their inverse rules are jungle rules in the sense of Definition 4.2.

2. The folding and unfolding rules given in 3.7 are jungle rules.

3. The following rule is a jungle rule:

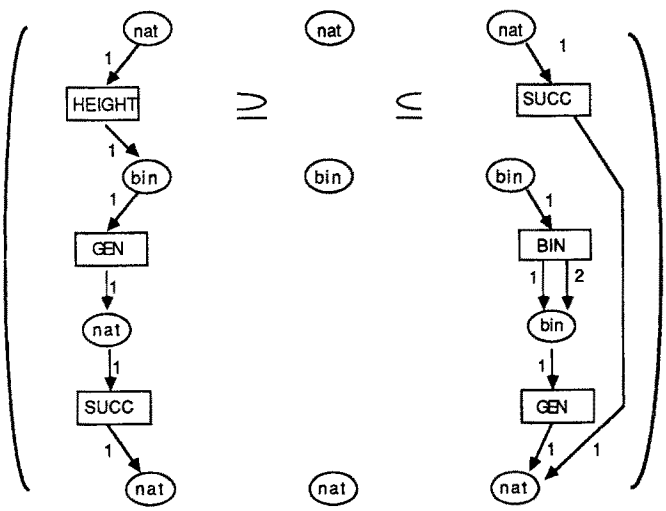


Figure 4.2: A Jungle Rule

4. Further jungle rules are shown in 5.4.

5. Note that the above definition is not symmetric, i.e., in general the inverse rule of a jungle rule is not a jungle rule.

6. The conditions of Definition 4.2 are necessary for jungle preservation in the following sense:
Let $r = (L \supseteq K \subseteq R)$ be a rule with jungles L, K, and R such that 4.2.1 or 4.2.2 is not satisfied. Then one can construct a jungle L' by adding one hyperedge to L (and extending SIG by one operation if necessary) such that $L' \underset{r}{\Rightarrow} R'$, where R' is not a jungle. □

[7] The relation $>_G$ on the nodes of a hypergraph G is defined for all $v_1, v_2 \in V_G$ by: $v_1 >_G v_2$ if and only if there is a non–empty path from v_1 to v_2. We write $v_1 \geq_G v_2$ if $v_1 >_G v_2$ or $v_1 = v_2$.

4.3 Jungle Preservation Theorem

Applying a jungle rule to a jungle yields a jungle.

Proof

Let G be a jungle, $r = (L \supseteq K \subseteq R)$ be a jungle rule, and $G \Rightarrow H$ be a direct derivation, given by the diagram

$$\begin{array}{ccccc} L & \supseteq & K & \subseteq & R \\ {\scriptstyle g}\downarrow & & {\scriptstyle d}\downarrow & & {\scriptstyle h}\downarrow \\ G & \supseteq & REM & \subseteq & GLUE \cong H \end{array}$$

It is sufficient to show that $GLUE$ is a jungle. Since REM and R are jungles, $GLUE$ satisfies condition 2.2.3 by construction. To show that the outdegree of each node in $GLUE$ is at most one, let $e_1, e_2 \in E_{GLUE}$ and $v \in V_{GLUE}$ such that $s_{GLUE}(e_1) = v = s_{GLUE}(e_2)$. For $v \in V_{REM} - V_{d(K)}$ we have $e_1, e_2 \in E_{REM}$, so $outdegree_{REM}(v) \leq 1$ implies $e_1 = e_2$. Analogously $v \in V_R - V_K$ implies $e_1, e_2 \in E_R$ and consequently $e_1 = e_2$ since $outdegree_R(v) \leq 1$. Assume therefore $v \in V_{d(K)}$.

Case 1: $e_1, e_2 \in E_R - E_K$. Let $v_1 = s_R(e_1)$ and $v_2 = s_R(e_2)$. Then $h_V(v_1) = v = h_V(v_2)$ so $v_1, v_2 \in V_K$. Since $v_1, v_2 \notin VAR_R$ there are $\overline{e_1}, \overline{e_2} \in E_L$ with $s_L(\overline{e_1}) = v_1$ and $s_L(\overline{e_2}) = v_2$, according to the definition of jungle rule. Further $\overline{e_1}$ and $\overline{e_2}$ must belong to $E_L - E_K$ as otherwise the outdegree of v_1 and v_2 in R would be greater than one. Now $s_G(g_E(\overline{e_1})) = g_V(v_1) = h_V(v_1) = v = h_V(v_2) = g_V(v_2) = s_G(g_E(\overline{e_2}))$ together with $outdegree_G(v) \leq 1$ implies $g_E(\overline{e_1}) = g_E(\overline{e_2})$. With the identification condition we conclude $\overline{e_1} = \overline{e_2}$. Hence $v_1 = v_2$ and, because $outdegree_R(v_1) \leq 1, e_1 = e_2$.

Case 2: $e_1 \in E_R - E_K, e_2 \in E_{REM}$. Analogously to the first case one shows that there is $\overline{e_1} \in E_L - E_K$ with $s_G(g_E(\overline{e_1})) = v$. But then $g_E(\overline{e_1}) \in E_{g(L)} - E_{g(K)}$ and $s_G(e_2) = v$ imply $outdegree_G(v) \geq 2$, contradicting the precondition that G is a jungle.

Case 3: $e_1, e_2 \in E_{REM}$. Then $s_{REM}(e_1) = s_{REM}(e_2)$ implies $e_1 = e_2$ since REM is a jungle.

Thus we have $outdegree_{GLUE}(v) \leq 1$ for all $v \in V_{GLUE}$.

It remains to show that $GLUE$ is acyclic. Suppose that $e_1, \ldots, e_n$ is a cycle in $GLUE$. Then some of these edges belong to $E_R - E_K$ since REM is acyclic. Further, because R is acyclic, at least one edge in the cycle has its source node in $V_{d(K)}$. Let a be an edge in $\{e_1, \ldots, e_n\} \cap (E_R - E_K)$. Since $outdegree_{GLUE}(s_{GLUE}(e_i)) = 1$ for $i = 1, \ldots, n$, a belongs to a path $a_1, \ldots, a_k$ in R from some $v \in V_K$ to some $x \in V_K \cap VAR_R$ such that $h_E(a_1), \ldots, h_E(a_k)$ is a subpath of $e_1, \ldots, e_n$. Therefore one can find pairs of nodes $(v_1, x_1), \ldots, (v_m, x_m)$ such that

(i) $v_i \in V_K$, $x_i \in VAR_R \cap V_K$, and $v_i >_R x_i$ for $i = 1, \ldots, m$,
(ii) $h_V(x_i) \geq_{REM} h_V(v_{i+1})$ for $i = 1, \ldots, m-1$ and $h_V(x_m) \geq_{REM} h_V(v_1)$.

We want to show $x_i \in VAR_L$ for $i = 1, \ldots, m$. Consider some $x \in \{x_1, \ldots, x_m\}$. $x \in VAR_R$ implies $outdegree_K(x) = 0$. Suppose that there is $e \in E_L - E_K$ with $s_L(e) = x$. By (i) and (ii) there is an edge in $GLUE$ with source $h_V(x)$. This edge cannot belong to REM as otherwise $s_G(g_E(e)) = g_V(x) = h_V(x)$ would imply $outdegree_G(g_V(x)) \geq 2$. Hence there is $e' \in E_R - E_K$ with $h_V(s_R(e')) = h_V(x)$. The definition of jungle rule implies that there is $\overline{e} \in E_L - E_K$ with $s_L(\overline{e}) = s_R(e')$. Consequently we have $s_G(g_E(e)) = h_V(x) = h_V(s_R(e')) = h_V(s_L(\overline{e})) = g_V(s_L(\overline{e})) = s_G(g_E(\overline{e}))$. Furthermore, e and $\overline{e}$ are different edges since $x \in VAR_R$ implies $s_L(e) = x \neq s_R(e') = s_L(\overline{e})$. With the identification condition we conclude $g_E(e) \neq g_E(\overline{e})$. Hence $outdegree_G(s_G(g_E(e))) \geq 2$, contradicting the assumption that G is a jungle.

Thus $x_1, \ldots, x_m \in VAR_L$ and we can rewrite (i) and (ii) to

(i') $v_i \in V_K$, $x_i \in VAR_L \cap V_K$, and $v_i >_R x_i$ for $i = 1, \ldots, m$,
(ii') $g_V(x_i) \geq_G g_V(v_{i+1})$ for $i = 1, \ldots, m-1$ and $g_V(x_m) \geq_G g_V(v_1)$.

Since r is a jungle rule, (i') implies $v_i >_L x_i$ for $i = 1, \ldots, m$. But then

$$g_V(v_1) >_G g_V(x_1) \geq_G g_V(v_2) >_G \ldots \geq_G g_V(v_m) >_G g_V(x_m) \geq_G g_V(v_1)$$

which contradicts the fact that G is acyclic.

□

5 Relating Term Rewriting and Jungle Evaluation

Jungle rules as introduced in the last section are general means for deriving jungles from jungles. In this section we define *evaluation rules* as special jungle rules which perform term rewriting on the terms of a given jungle. In general, the application of an evaluation rule to a jungle corresponds to a set of sequences of rewrite steps, performed on certain terms in that jungle.

We assume familiarity with the notion of term rewriting as defined, for example, by Huet and Oppen [HO 80] or Klop [Kl 87].

5.1 General Assumption

Let TR be a term rewriting system in which each rewrite rule $l \to r$ consists of two terms l, r of equal sort such that l is not a variable and the variables in r occur already in l. Moreover, we assume that r is not a variable.

Notation

The rewrite relation associated with TR is denoted by $\to$. We write $\overset{*}{\to}$ and $\overset{n}{\to}$ for the reflexive transitive closure and the n-fold composition of $\to$, respectively. □

5.2 Preliminary Considerations

Given a rewrite rule $l \to r$, we are searching for jungle rules corresponding to the rewrite rule. For handling this problem, one may argue as follows.

(1) Choose $jungle(l)$ as the left–hand side of the rule. $jungle(l)$ (more precisely, the root of $jungle(l)$) represents the term l. Moreover, among all l–representing jungles with one root, $jungle(l)$ is "largest", i.e., for each jungle J with one root representing l there is a surjective hypergraph morphism $fold : jungle(l) \to J$. Whenever J has an occurrence in a jungle G, $jungle(l)$ has an occurrence, too. The converse is not true. Thus, $jungle(l)$ is the most attractive candidate.

(2) Choose $JUNGLE(r)$ as the right–hand side of the rule. $JUNGLE(r)$ (more precisely, the root of $JUNGLE(r)$) represents the term r. Moreover, among all r–representing jungles (with one root) $JUNGLE(r)$ is the most efficient representation.

(3) Finally, we have to choose a gluing jungle. For this we may choose the root of $jungle(l)$ which corresponds to the root of $JUNGLE(r)$ as well as the variable nodes of $JUNGLE(r)$. (Note that by Assumption 5.1 $VAR(r) \subseteq VAR(l)$ and hence the variable nodes of $JUNGLE(r)$ are contained in $jungle(l)$).

In this way, one gets a rule corresponding to the given rewrite rule which – unfortunately – does not yet work well. Where is the problem?

Let $l \to r$ be a rewrite rule and t be a term. Let $p = (L \supseteq K \subseteq R)$ be the corresponding rule constructed according to (1) – (3) and T be an arbitrary jungle representing t. Since in jungles the indegree of a node is not bounded by 1, $l \to r$ may be applicable to t while the corresponding rule p is not applicable to T (because the contact condition may not hold).

To avoid this problem as far as possible we proceed as follows: We choose L as described in (1). In contrast to (2) and (3) we do not delete the jungle L (up to the root and some variable nodes), but preserve $L - \{e\}$ where e denotes the hyperedge outcoming from the root of L. Thus, in the new rule all nodes of L remain preserved by an application of the rule. □

5.3 Definition (Evaluation Rule and Evaluation Step)

1. Let $l \to r$ be a rewrite rule in TR. A jungle rule $p = (L \supseteq K \subseteq R)$ is called an *evaluation rule for* $l \to r$ if the following conditions are satisfied:

1. L is the variable-collapsed tree $jungle(l)$,
2. $K = L - \{e\}$ [8] where e denotes the hyperedge outgoing from $root_L$,

[8] For a hypergraph L and a hyperedge $e \in E_L$, $L - \{e\}$ denotes the subhypergraph X with $V_X = V_L$ and $E_X = E_L - \{e\}$.

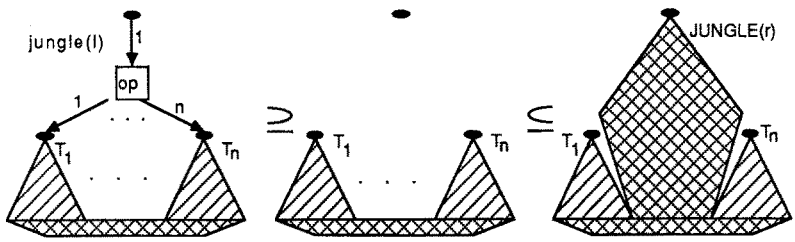
3. R is a jungle containing K such that $term_R(root_L) = r$.

$l \to r$ is said to be the *underlying rewrite rule* of p.

2. A direct derivation $G \Rightarrow H$ through an evaluation rule is called an *evaluation step* provided that G (and thus H) is a jungle. □

5.4 Remarks, Illustrations, and Examples

1. There is a (non-empty) set of evaluation rules for each rewrite rule $l \to r$:
By assumption, l possesses at least one operation symbol. Thus, there is a unique hyperedge e in $L = jungle(l)$ outgoing from the root and corresponding to the leftmost operation symbol in l. Only this hyperedge does not occur in the gluing jungle $K = jungle(l) - \{e\}$. The requirement for the right-hand side R is more liberal then that for L and K. There are several (non-isomorphic) jungles which may be chosen as right–hand side of the rule. For instance, $JUNGLE(r)$ combined with the gluing jungle K is a candidate. The obtained rule may be depicted as follows.



The rule $(L \supseteq K \subseteq R)$ turns out to be a jungle rule in the sense of Definition 4.2 because (1) $VAR_L = VAR_R$ and (2) $root_L >_L x$ for all $x \in VAR_L$ and, for all $v \in V_K - \{root_L\}$ and all $x \in VAR_L$, $v >_R x$ if and only if $v >_L x$.

2. Consider the rewrite rules of the **generate** specification (given in the introduction):
$GEN(0) \to BIN(EMPTY, EMPTY)$ and $GEN(SUCC(N)) \to BIN(GEN(N), GEN(N))$. Corresponding evaluation rules are shown in Figure 5.1.

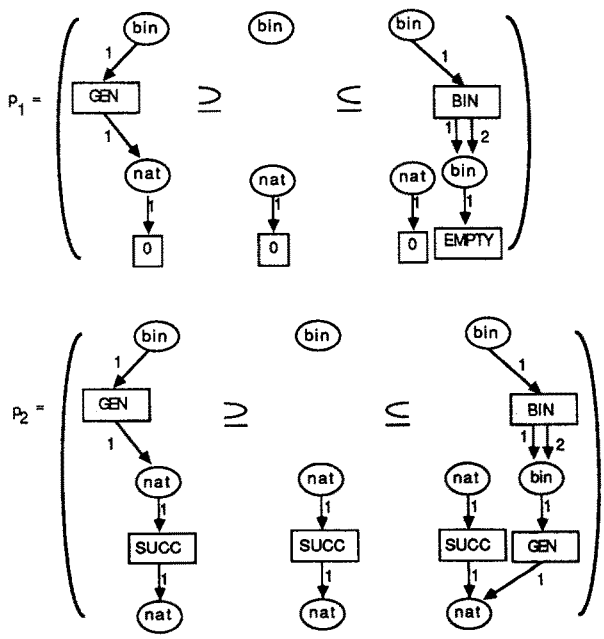


Figure 5.1: Evaluation Rules

3. Evaluation rules as defined in 5.3 do not remove nodes. Thus, for each evaluation step $G \Rightarrow H$, the nodes of G are contained in H up to renaming. For notational convenience we will ignore the renaming and assume that $V_G \subseteq V_H$. □

In order to check the applicability of an evaluation rule to a given jungle one just has to find an occurrence of the left-hand side in that jungle.

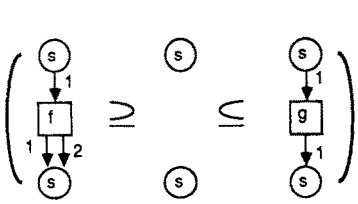
5.5 Lemma (Applicability of Evaluation Rules)
Let $p = (L \supseteq K \subseteq R)$ be an evaluation rule and let G be a jungle. Then p is applicable to G (i.e., there is an evaluation step $G \underset{p}{\Rightarrow} H$) if and only if there is a hypergraph morphism $g : L \to G$.

Proof
By the definition of direct derivation, $G \underset{p}{\Rightarrow} H$ implies that there is a hypergraph morphism from L to G. Conversely, let $g : L \to G$ be a hypergraph morphism. The contact condition is satisfied since $V_K = V_L$. To show that g satisfies also the identification condition, let e be the unique hyperedge with $s_L(e) = root_L$ and let e' be any hyperedge in L different from e. Then $s_L(e) >_L s_L(e')$ by the structure of L so $g_V(s_L(e)) >_G g_V(s_L(e'))$. Hence $g_V(s_L(e)) \neq g_V(s_L(e'))$ according to the acyclicity of G. Consequently we have $g_E(e) \neq g_E(e')$, i.e., g satisfies the identification condition because each two other items belong to the gluing part and hence may be identified. Thus there is a direct derivation $G \underset{p}{\Rightarrow} H$ based on g. □

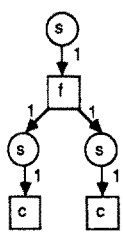
Remark
By Lemma 5.5, an evaluation rule $(L \supseteq K \subseteq R)$ is applicable to a jungle G whenever L has an occurrence in G. Nevertheless, the problem mentioned in 5.2 is not solved completely. Let us explain this in more detail. A nice result would be the following:

Let $l \to r$ be an arbitrary rewrite rule and t be a term. Moreover, let p be an evaluation rule for $l \to r$ and T be a jungle representing t. Then p is applicable to T provided that $l \to r$ is applicable to t.

Unfortunately, in the case that l is a term in which a variable occurs several times this is not true for all jungles T representing t. Consider, for example, the rewrite rule $f(x,x) \to g(x)$ which is applicable to $f(c,c)$. Then the evaluation rule



is not applicable to



On the other hand, it is applicable to the fully collapsed jungle $JUNGLE(f(c,c))$. What is to be done?

(1) If all rewrite rules are *left-linear*, i.e., in the left-hand side of the rule each variable occurs only once, the problem does not occur. But left-linear term rewriting systems seem to be too restrictive.

(2) If the considered jungles are "folded" (i.e. suitably collapsed) the problem does not occur. Thus, our proposal for handling the problem is to use the folding rules introduced in Section 3 to compress jungles (if necessary) before applying evaluation rules. □

5.6 Example
The evaluation rule p_2 given in Figure 5.1 can be applied to the jungle G yielding the jungle H given in

Figure 5.2.

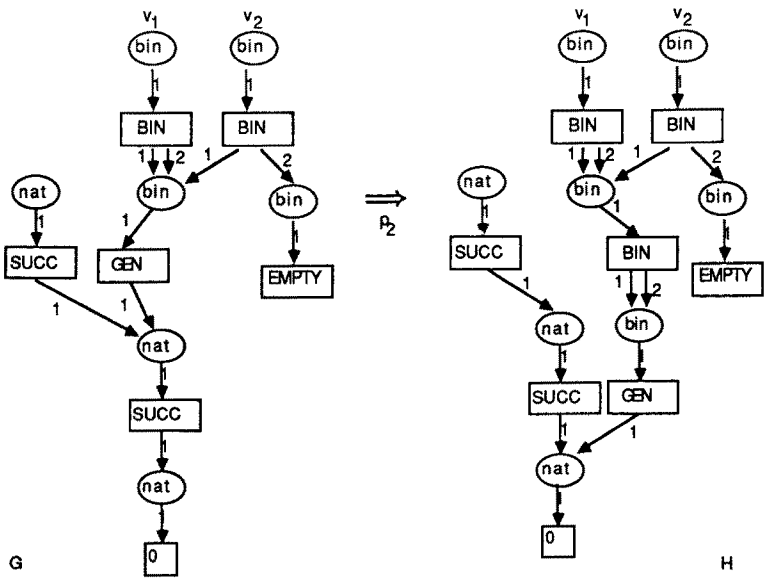


Figure 5.2: Application of an Evaluation Rule

The jungle G represents a collection of terms, for example
$term_G(v_1) = BIN(GEN(SUCC(0)), GEN(SUCC(0)))$ and
$term_G(v_2) = BIN(GEN(SUCC(0)), EMPTY)$.
Concerning these nodes, the evaluation step through the evaluation rule p_2 performs the rewriting steps $BIN(GEN(SUCC(0)), GEN(SUCC(0))) \xrightarrow{2} BIN(BIN(GEN(0), GEN(0)), BIN(GEN(0), GEN(0)))$ and $BIN(GEN(SUCC(0)), EMPTY) \xrightarrow{1} BIN(BIN(GEN(0), GEN(0)), EMPTY)$. □

The application of an evaluation rule to a jungle rewrites the term represented by the root of the occurrence of the left-hand side. Simultaneously, the terms of all nodes from which there is a path to this root are rewritten, too. Moreover, the existence of several distinct paths leads to a sequence of rewrite steps.

5.7 Theorem (Evaluation Steps Perform Term Rewriting)
Let $G \Rightarrow H$ be an evaluation step through an evaluation rule $(L \supseteq K \subseteq R)$ with an occurrence $g : L \to G$. Then for each $v \in V_G$, $term_G(v) \xrightarrow{n} term_H(v)$ where n is the number of paths from v to $g_V(root_L)$.

Sketch of Proof
The occurrence g induces a substitution $\sigma : T_{SIG}(VAR_L) \to T_{SIG}(VAR_G)$ satisfying $term_G(g_V(v)) = \sigma(term_L(v))$ for each $v \in V_L$. In particular we have $term_G(g_V(root_L)) = \sigma(term_L(root_L)) = \sigma(l)$ for the left-hand side l of the underlying rewrite rule $l \to r$. By the definition of evaluation rule one gets $term_H(g_V(root_L)) = \sigma(term_R(root_L)) = \sigma(r)$ so $term_G(g_V(root_L)) \to term_H(g_v(root_L))$. Now let $v \in V_G$ such that there are n paths from v to $g_V(root_L)$ for some $n \geq 1$. Then there are n distinct occurrences of the subterm $\sigma(l)$ in $term_G(v)$ which are simultaneous rewritten to $\sigma(r)$ by the evaluation step, so $term_G(v) \xrightarrow{n} term_H(v)$ holds. Finally we have $term_G(v) \xrightarrow{0} term_H(v)$ for all nodes v in G from which there is no path to $g_V(root_L)$, since these nodes represent in H the same terms as in G. □

The situation described in Theorem 5.7 may be depicted as follows:

$$\begin{array}{ccccc} & & G & \overset{\Longrightarrow}{\underset{EVAL}{}} & H & & \\ \text{REPRESENTS} & & \downarrow & & \downarrow & & \text{REPRESENTS} \\ & & t & \overset{*}{\underset{TR}{\longrightarrow}} & u & & \end{array}$$

where $EVAL$ denotes a set of evaluation rules for TR.

In general, one evaluation step corresponds to a set of sequences of rewrite steps as stated in Theorem 5.7. Nevertheless, each single rewrite step can be simulated by an evaluation step if an appropriate jungle is chosen for representing the term to be rewritten.

5.8 Theorem (Simulation of Rewrite Steps)
Let $t \to u$ be a rewrite step. Then there is an evaluation step $G \Rightarrow H$ where G contains a node v_0 satisfying $term_G(v_0) = t$ and $term_H(v_0) = u$.

Proof
Let $l \to r$ and σ be the rewrite rule and the substitution associated with the rewrite step $t \to u$. Let T be the fully collapsed tree with $term_T(root_T) = \sigma(l)$. By a derivation $T \overset{*}{\underset{GEN}{\Longrightarrow}} G$ one can construct a jungle G containing T and a node v_0 such that $term_G(v_0) = t$ and

$(*)$ changing $term_G(root_T)$ from $\sigma(l)$ to $\sigma(r)$ yields $term_G(v_0) = u$.[9]

Now let $p = (L \supseteq K \subseteq R)$ be an evaluation rule for $l \to r$. Then for each $v \in V_L$ there is exactly one $v' \in V_T$ with $term_G(v') = \sigma(term_L(v))$. This correspondence uniquely extends to a hypergraph morphism $g : L \to G$ satisfying $term_G(g_V(v)) = \sigma(term_L(v))$ for each $v \in V_L$. Hence there is an evaluation step $G \overset{}{\underset{p}{\Longrightarrow}} H$ based on g, according to Lemma 5.5. Further $g_V(root_L) = root_T$ holds, so $term_H(root_T) = \sigma(r)$ by the definition of evaluation rule. Now $(*)$ implies $term_H(v_0) = u$. □

The situation described in Theorem 5.8 may be depicted as follows.

$$\begin{array}{ccccc} & & t & \overset{}{\underset{TR}{\longrightarrow}} & u & & \\ \text{REPRESENTS} & & \uparrow & & \uparrow & & \text{REPRESENTS} \\ & & G & \overset{\Longrightarrow}{\underset{EVAL}{}} & H & & \end{array}$$

Moreover, if PRE denotes the set of folding and unfolding rules given in 3.7, one gets that for each rewrite step $t \to u$ and each jungle $\overline{G}$ representing t there are derivations $\overline{G} \overset{*}{\underset{PRE}{\Longrightarrow}} G$ and $G \overset{}{\underset{EVAL}{\Longrightarrow}} H$ such that H represents u. In this sense, term rewriting may be simulated by jungle evaluation.

6 Discussion

Based on the fundamental study in [Pl 86], we have introduced jungle evaluation as a certain type of graph rewriting in this paper. As jungles are inductively structured and each one represents a set of terms over a signature in a neat and compact way, the main emphasis has been put on jungle preservation. We have specified a class of graph rewrite rules which yield jungles whenever they are applied to jungles. Moreover, jungle evaluation has been related to algebraic specifications. We have demonstrated how jungle rewriting can simulate term rewriting. There is some evidence that the use of jungles rather than terms may speed up the evaluation process drastically.

This paper presents a first step in the systematic investigation of jungle evaluation. More work will have to be done to realize the potential for this new graph grammar approach to the evaluation of functional expressions. We would like to point out some topics for future investigation.

[9] I.e., there is exactly one path from v_0 to $root_T$ and this path corresponds, when viewing $t \to u$ as a subtree replacement, to the path from the root of t to the node where $\sigma(l)$ is replaced by $\sigma(r)$.

(1) The prospect of efficiency: Although the sizes of terms may be multiplied in each term rewrite step, the sizes of jungles never grow beyond a constant bound in a single rewrite step. Furthermore, a jungle rewrite step can correspond to several term rewrite steps. For which examples and under which conditions do these facts pay off? We strongly believe that jungle evaluation is more efficient than term evaluation. But how can this be stated and proved in a precise way?

(2) The prospect of parallelism: The theory of graph grammars provides some concepts for parallelism and concurrency which apply to jungle rewriting. But how can this knowledge be exploited in a meaningful and advantageous way? Does this kind of parallelism help to speed up the evaluation process once more?

(3) The prospect of a new specification and programming paradigm: So far, jungle evaluation is discussed as a method for optimizing algebraic specifications in certain respects. But why should jungle rewrite rules not be interesting in their own right? There are excellent arguments for algebraic and equational specification and programming based on terms and equations. Are there good arguments for a new specification and programming paradigm based on jungles and jungle rewrite rules as well?

7 Appendix: Hypergraphs and Hypergraph Replacements

This appendix summarizes all notions concerning hypergraphs, generalizing the corresponding notions for graphs, and hypergraph replacements which are used in this paper.

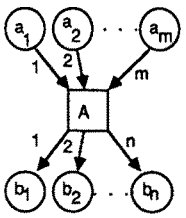
7.1 Definition (Hypergraph)
1. Let $C = (C_V, C_E)$ be a pair of sets, called a pair of *label alphabets* for nodes and hyperedges respectively, which will be fixed in the following.

2. A *hypergraph* (over C) is a system $H = (V_H, E_H, s_H, t_H, l_H, m_H)$, where V_H is a finite set of *nodes*, E_H is a finite set of *hyperedges* (or *edges* for short), $s_H : E_H \to V_H^*$ and $t_H : E_H \to V_H^*$ [10] are two mappings assigning a sequence of *sources* $s_H(e)$ and a sequence of *targets* $t_H(e)$ to each $e \in E_H$, and $l_H : V_H \to C_V$ and $m_H : E_H \to C_E$ are two mappings assigning *labels* to nodes and hyperedges.

3. The set of all hypergraphs (over C) is denoted by $\mathcal{H}_C$.

4. Let $H \in \mathcal{H}_C, e \in E_H, s_H(e) = x_1 \ldots x_m$ and $t_H(e) = x_{m+1} \ldots x_{m+n}$ with $x_i \in V_H$ for $i = 1, \ldots, m+n$. Then $NODES_H(e) = \{x_i \in V_H \mid i = 1, \ldots, m+n\}$ denotes the set of nodes related by the hyperedge e.

Remarks
1. In drawings of hypergraphs, a circle represents a node and the label is inscribed in the circle; a graphical structure of the form



depicts a hyperedge with sources and targets where the label of the hyperedge is inscribed in the box, the i-th arrow incoming into the box starts at the i-th source (i=1,...,m) and the j-th arrow outgoing from the box reaches the j-th target (j=1,...,n). In other words, our graphical representation makes use of the one-to-one correspondence between hypergraphs and bipartite graphs.

2. There is a one-to-one correspondence between hypergraphs and bipartite graphs: On one hand, for each hypergraph H there is an underlying graph $U(H)$, whose nodes are the nodes as well as the hyperedges of H and whose edges are the "tentacles" of the hyperedges of H. On the other hand, each (directed) bipartite graph G may be seen as a hypergraph choosing the one set of nodes as the set of nodes and the other set

[10] Let A be a set. Then A^* denotes the set of finite sequences over A, including the empty sequence λ.

as the set of hyperedges. This close relationship between hypergraphs and bipartite graphs is interesting in several respects: (1) Graph-theoretic notions can be adapted to hypergraphs. (2) Various known results on graph grammars can be applied to hypergraph grammars (see, i.e., [EK 76], [Eh 79], [Kr 87]).

7.2 Definition (Subhypergraph and Hypergraph Morphism)
1. Let $G, H \in \mathcal{H}_C$. Then G is called a *subhypergraph* of H, denoted by $G \subseteq H$, if $V_G \subseteq V_H, E_G \subseteq E_H$ and s_G, t_G, l_G, and m_G are restrictions of the corresponding mappings in H.

2. Let $G, H \in \mathcal{H}_C$ and $V \subseteq V_H$. Then G is said to be the *subhypergraph of H induced by V* if $G \subseteq H$, $V_G = V$, and $E_G = \{e \in E_H \mid NODES_H(e) \subseteq V\}$.

3. Let $G, H \in \mathcal{H}_C$. A *hypergraph morphism* $h : G \to H$ consists of two mappings $h_V : V_G \to V_H$ and $h_E : E_G \to E_H$ such that $s_H(h_E(e)) = h_V^*(s_G(e))$, $t_H(h_E(e)) = h_V^*(t_G(e))$,[11] $l_H(h_V(v)) = l_G(v)$, and $m_H(h_E(e)) = m_G(e)$ for all $e \in E_G$, $v \in V_G$. h is called *injective* (*surjective*, *bijective*) if both h_V and h_E are injective (surjective, bijective).

4. A bijective hypergraph morphism $h : G \to H$ is called an *isomorphism*. In this case, G and H are said to be *isomorphic*, denoted by $G \cong H$.

Remark
Each hypergraph morphism $h : G \to H$ determines a subhypergraph of H, denoted by $h(G)$.

7.3 Definition (Hypergraph Rule)
A (*gluing*) *rule* $r = (L \supseteq K \subseteq R)$ consists of three hypergraphs L, R and K with $K \subseteq L$ and $K \subseteq R$. L is called the *left-hand side*, R the *right-hand side*, and K the *gluing hypergraph* of r.

7.4 Definition (Hypergraph Derivation)
1. Let $G, H \in \mathcal{H}_C$ and $r = (L \supseteq K \subseteq R)$ be a rule. Then G *directly derives H through r*, if H is isomorphic to the hypergraph $GLUE$ constructed in the following four steps:

1. *CHOOSE* a hypergraph morphism $g : L \to G$, called the (*left-*)*occurrence map.*
2. *CHECK* the following *gluing condition* (consisting of the *contact condition* and the *identification condition*):
   - For all $v \in V_L$, if $g_V(v) \in NODES_G(e)$ for some $e \in E_G - E_{g(L)}$, then $v \in V_K$.
   - For all $x, y \in L$,[12] if $x \neq y$ but $g(x) = g(y)$, then $x, y \in K$.
3. *REMOVE* the occurrence $g(L)$ of L in G up to $g(K)$ from G yielding the *remainder* $REM = G - (g(L) - g(K))$.[13]
4. *ADD* the right-hand side R up to K to REM yielding the *gluing* $GLUE = REM + (R - K)$ [14] the components of which are defined as follows:
   - $V_{GLUE} = V_{REM} + (V_R - V_K)$ and $E_{GLUE} = E_{REM} + (E_R - E_K)$,
   - each node and each hyperedge keeps its label,
   - each hyperedge of E_{REM} keeps its sources and targets,
   - each hyperedge of $E_R - E_K$ keeps its sources and targets provided that they are in $V_R - V_K$; otherwise sources and targets are handed over to V_{REM}: $s_{GLUE}(e) = h_V^*(s_R(e))$ and $t_{GLUE}(e) = h_V^*(t_R(e))$ for $e \in E_R - E_K$ where $h_V : V_R \to V_{GLUE}$ is given by $h_V(v) = v$ for $v \in V_R - V_K$ and $h_V(v) = g_V(v)$ for $v \in V_K$.

[11] Let $f : A \to B$ be a mapping. Then $f^* : A^* \to B^*$ denotes the free symbolwise extension of f given by $f^*(a_1 \ldots a_k) = f(a_1) \ldots f(a_k)$ for all $k \in \mathcal{N}$ and $a_i \in A (i = 1, \ldots, k)$.

[12] "$x \in L$" is a short denotation for "$x \in V_L$ or $x \in E_L$" and is used whenever nodes and hyperedges do not have to be destinguished. Similarly, "$g(x)$" is short for "$g_V(x)$ or $g_E(x)$".

[13] " $-$ " denotes the difference of sets for nodes and hyperedges separately.

[14] " $+$ " denotes the disjoint union of sets for nodes and hyperedges separately.

Such a *direct derivation* is denoted by $G \Rightarrow H$ *through* r (*based on* g) or $G \underset{r}{\Rightarrow} H$.

2. A sequence of direct derivations of the form $H_0 \underset{r_1}{\Rightarrow} H_1 \underset{r_2}{\Rightarrow} \ldots \underset{r_k}{\Rightarrow} H_k$ constitutes a *derivation from* H_0 *to* H_k *through* $r_1, \ldots, r_k$. Such a derivation may be denoted by $H_0 \underset{RULES}{\overset{*}{\Longrightarrow}} H_k$ if $r_1, \ldots, r_k \in RULES$.

Remarks

1. The occurrence map $g : L \to G$ locates the *occurrence* $g(L)$ *of* L *in* G.

2. The gluing condition ensures that REM is a subhypergraph of G.

3. Let $REM = G - (g(L) - g(K))$. Then, for $x \in K$, $d(x) = g(x)$ defines a hypergraph morphism $d : K \to REM$ locating $g(K)$ in REM.

4. Let H be isomorphic to $GLUE$ and $i : GLUE \to H$ be the corresponding isomorphism. Then $h(x) =$ **if** $x \in R - K$ **then** $i(x)$ **else** $i(d(x))$ defines a hypergraph morphism $h : R \to H$, called the *right-occurrence map*. Respectively, $h(R)$ is called the *occurrence of* R *in* H. Further the restriction of $i : GLUE \to H$ to $REM \subseteq GLUE$ defines a hypergraph morphism.

5. All hypergraph morphisms involved in a direct derivation can be grouped into two squares

$$\begin{array}{ccccc} L & \leftarrow & K & \rightarrow & R \\ g\downarrow & & d\downarrow & & h\downarrow \\ G & \leftarrow & REM & \rightarrow & H \end{array}$$

which are actually gluing diagrams in the sense of [Eh 79], 2.6. The explicit construction of the derived hypergraph given above corresponds to the gluing analysis and the gluing construction in [Eh 79], 3.7 and 3.8.

The definition of a direct derivation is symmetric in the following sense.

7.5 Fact

Let $G \Rightarrow H$ be a direct derivation through $r = (L \supseteq K \subseteq R)$ based on the (left–)occurrence map $g : L \to G$. Let $h : R \to H$ be the corresponding right-occurrence map. Then there is a direct derivation $H \Rightarrow G$ through the *inverse rule* $r^{-1} = (R \supseteq K \subseteq L)$ based on the occurrence map $h : R \to H$.

Acknowledgment

Wolfgang Ditt, Inger Kuhlmann, Anne Wilharm, and the three authors of this paper took part in a weekend seminar on graph grammars held at the second author's private home in Summer 1985. With a lot of fun, the six of us developed the idea of jungle evaluation and outlined the basic notions and expected results. As the authors of this paper, we gratefully acknowledge Anne's, Inger's, and Wolfgang's contribution to the initial seminar.

Moreover, we would like to thank Don Sannella, Michael Löwe, and the anonymous referees for many helpful comments on the draft of this paper.

References

[BEGKPS 87] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, M.R. Sleep: *Term Graph Reduction.* University of East-Anglia and University of Nijmegen, Proc. PARLE Conference on Parallel Architectures and Languages, Eindhoven (1987).

[Eh 79] H. Ehrig: *Introduction to the Algebraic Theory of Graph Grammars.* Proc. 1st Graph Grammar Workshop, Lect. Not. Comp. Sci. 73, 1-69 (1979).

[EK 76] H. Ehrig, H.-J. Kreowski: *Parallelism of Manipulations in Multidimensional Information Structures.* Proc. MFCS'76, Lect. Not. Comp. Sci. 45, 284-293 (1976).

[EM 86] H. Ehrig, B. Mahr: *Fundamentals of Algebraic Specifications. Part I: Initial Semantics.* Springer, Monographs in Computer Science, New York-Berlin-Heidelberg (1986).

[ER 76] H. Ehrig, B.K. Rosen: *Commutativity of Independent Transformations on Complex Objects.* Research Report RC 6251. IBM T.J. Watson Research Center, Yorktown Heights (1976).

[HO 80] G. Huet, D.C. Oppen: *Equations and Rewrite Rules, A Survey.* In R.V. Book (ed.): *Formal Language Theory: Perspectives and Open Problems.* Academic Press, 349-405 (1980).

[Kl 87] J.W. Klop: *Term Rewriting Systems. A Tutorial.* EATCS Bulletin 32, 142-182 (1987).

[Kr 87] H.-J. Kreowski: *Is Parallelism Already Concurrency? – Part 1: Derivations in Graph Grammars.* Proc. 3rd Graph Grammar Workshop, Lect. Not. Comp. Sci. 291, 343-360 (1987).

[Pa 82] P. Padawitz: *Graph Grammars and Operational Semantics.* Theor. Comp. Sci. 19, 117-141 (1982).

[Pl 86] D. Plump: *Im Dschungel: Ein neuer Graph–Grammatik–Ansatz zur effizienten Auswertung rekursiv definierter Funktionen.* Diplomarbeit, Studiengang Informatik, Universität Bremen (1986).

[St 80] J. Staples: *Computation on Graph–like Expressions.* Theor. Comp. Sci. 10, 171-185 (1980).