Proceedings Paper:
Plump, DETLEF and Hoffmann, Berthold (1988) Jungle Evaluation for Efficient Term Rewriting. In: Algebraic and Logic Programming, International Workshop (ALP 1988). Lecture Notes in Computer Science. Akademie-Verlag Berlin, pp. 191-203.

https://doi.org/10.1007/3-540-50667-5_71

Jungle Evaluation for Efficient Term Rewriting *

Berthold Hoffmann and Detlef Plump **

Abstract

Jungles are acyclic hypergraphs that represent sets of terms over a signature so that equal subterms can be shared.

Term rewrite rules can be translated into jungle evaluation rules having the property that each evaluation step performs several term rewrite steps simultaneously.

It is shown that the translation preserves termination. Confluent and terminating term rewriting systems are translated into confluent and terminating jungle evaluation systems which are complete in the sense that each term normal form is represented by some jungle normal form.

Moreover, jungle evaluation can be speeded up by additional hypergraph rules that fold equal terms without loosing termination and confluence. Using these folding rules, even non-linear term rewriting systems can be modelled.

1 Introduction

Term rewriting is an interesting way of "computing by replacement" which is used in various areas of computer science: For the interpretation of functional and logical programming languages (cf. [Der 85], [O'D 87], [Pey 87]), for theorem proving (cf. [Hsi 85]), and for executing algebraic specifications of abstract data types and checking properties of specifications like consistency and completeness (cf. [EM 85], [FGJM 85], [Pad 83]).

In classical implementations terms are represented by *trees*, and rewriting is realized by *subtree replacement*. Unfortunately, this way of rewriting may be very expensive, both in time and space: The application of a rule may require large subterms to be copied, and for each copy of a term, the result must be computed anew.

In this paper, we investigate an improved model for implementing term rewriting where these sources of inefficiency are avoided:

- Terms are represented by acyclic hypergraphs rather than by trees. These hypergraphs are called *jungles*, a name coined in [Plu 86] and [HKP 88]. In jungles, multiple occurrences of terms can be shared.
- Rewriting is performed by hypergraph replacement, specified by *evaluation rules* according to the algebraic theory of graph grammars (see, e.g., [Ehr 79]). By applying these evaluation rules, new references to existing subterms are introduced instead of copying subterms completely.
- Additional hypergraph rules for *folding* multiple occurrences of subterms allow each term in a jungle to be represented uniquely, so that its result is computed at most once.

1.1 Example (Computation of Fibonacci Numbers)
Consider the term rewrite rules

$$\begin{aligned} \texttt{fib(0)} &\to \texttt{0} \\ \texttt{fib(succ(0))} &\to \texttt{succ(0)} \\ \texttt{fib(succ(succ(x)))} &\to \texttt{fib(succ(x))} + \texttt{fib(x)} \end{aligned}$$

*This work is supported by the Commission of the European Communities under Contract 390 (PROSPECTRA Project) in the ESPRIT Programme.

** Fachbereich Mathematik und Informatik, Universität Bremen, Postfach 330 440, D-2800 Bremen 33. Usenet: {hof,det}%informatik.uni-bremen.DE

specifying a function fib that computes fibonacci numbers, based on natural numbers with the constant 0, successor function succ, and addition +.

For computing the fibonacci number of 4, term rewriting starts as follows:

$$\begin{aligned} \mathtt{fib}(\mathtt{succ}^4(0)) &\rightarrow \mathtt{fib}(\mathtt{succ}^3(0)) + \mathtt{fib}(\mathtt{succ}^2(0)) \\ &\rightarrow \mathtt{fib}(\mathtt{succ}^2(0)) + \mathtt{fib}(\mathtt{succ}(0)) + \mathtt{fib}(\mathtt{succ}^2(0)) \ldots \end{aligned}$$

In both steps, the arguments of fib are copied. The resulting term contains two copies of $\mathtt{fib}(\mathtt{succ}^2(0))$; each of them must be rewritten anew. As a consequence, rewriting a term $\mathtt{fib}(\mathtt{succ}^n(0))$ to normal form requires time and space exponential in n.

Figure 1 shows the corresponding jungle evaluation steps, followed by one folding step. After the folding step, $\mathtt{fib}(\mathtt{succ}^2(0))$ and its subterms are represented just once in the jungle, and thus have to be evaluated at most once. By performing evaluation and folding steps in this order, the rewriting of a term $\mathtt{fib}(\mathtt{succ}^n(0))$ to normal form requires only a number of steps, and space linear in n (see section 4). □

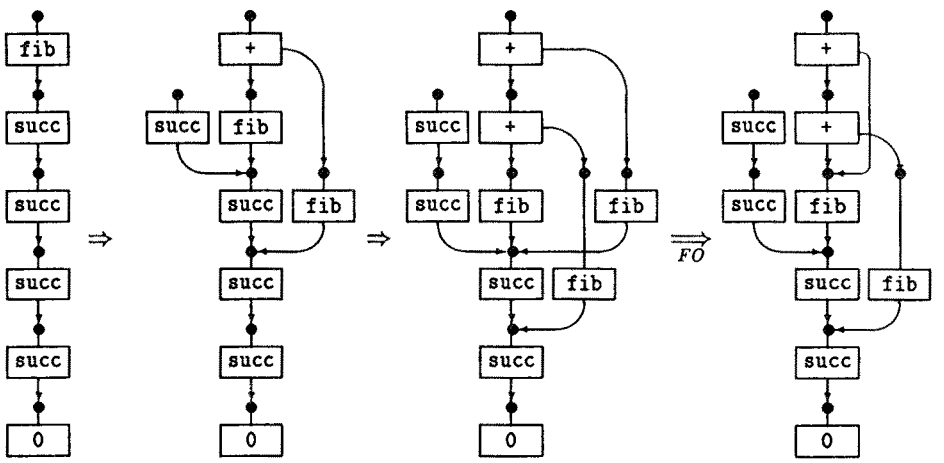


Figure 1: Jungle evaluation steps followed by a folding step

The first graph grammar approach to the evaluation of functional expressions has been undertaken by Ehrig, Rosen, and Padawitz ([ER 76], [Pad 82]). However, they use an extension of the algebraic theory of graph grammars having the drawback that most of the results for "standard graph grammars" are not applicable. The graph reduction approaches of Staples [Sta 80] and Barendregt et. al. [BEGKPS 87] use formalisms of graph rewriting for which no general theory is available.

In [Hof 83] and [Plu 86], the authors started an approach to model term rewriting by standard graph grammars. H-J. Kreowski, A. Habel, and D. Plump continued this work in [HKP 88], where hypergraphs are used instead of graphs, in order to make the technical treatment easier.

With this paper, we carry forth this research by using a slightly more general class of hypergraph grammars which allows for a translation of term rewrite rules into evaluation rules without need for "indirect pointers" or additional identity operations. We investigate general conditions for *termination* and *confluence* of jungle evaluation.

The paper is organised as follows:

In section 2, we recall the definition of jungles and state basic results about the relationship between jungles and terms. Rules for folding multiple representations of terms are introduced which allow to compute minimal jungle representations for sets of terms. In section 3, we construct jungle evaluation rules for modelling term rewriting. It is shown that these rules perform multiple parallel term rewrite steps and compute normal forms with respect to term rewriting. In particular, it is demonstrated that termination of term rewriting carries over to jungle evaluation. The corresponding result for confluence holds if the given term rewriting system is terminating. In section 4, we make some preliminary observations concerning the efficiency of our model of term rewriting and discuss briefly open problems and possible applications of our approach.

In the appendix, hypergraphs and hypergraph derivations are reviewed.

We assume familiarity with basic notions of term rewriting as signature, term, substitution, subterm replacement, etc. (see for instance [EM 85], [HO 80], [Klo 87]). All proofs omitted in this paper can be found in the technical report [HP 88].

Acknowledgements
We would like to thank H.-J. Kreowski and our colleagues from the PROSPECTRA team at Universität Bremen for providing valuable feedback. We are indebted to Clemens Lautemann who provided the key idea for the proof of the termination theorem 3.10. Special thanks are due to Annegret Habel for numerous fruitful discussions and many suggestions improving the presentation.

2 Jungles and Their Relationship to Terms

To realize sharing of common subterms, terms have to be represented by graph-like structures. We use the "Berlin-approach" of graph grammars (see e.g. [Ehr 79]) as a formal framework for the description of graph transformations. However, in the papers of Ehrig, Rosen, and Padawitz on function evaluation ([ER 76], [Pad 82]) this approach is modified in order to allow certain changes of node labels. Our intention is to avoid these modifications to have the results of the "Berlin approach" available.

We represent operation symbols as edges rather than as nodes. Under this requirement functional expressions are more adequately represented by hypergraphs than by graphs since an operation symbol of arity n can be represented as one hyperedge whereas in the graph case n+1 edges together with a node are needed.

The adaptation of notions and results of graph grammar theory to the hypergraph case is straightforward due to the one-to-one correspondence between bipartite graphs and hypergraphs. The basic notions concerning hypergraphs and hypergraph derivations are summarized in the appendix.

This chapter starts with the definition of jungles given in [HKP 88] and shows in which way jungles represent terms. For each jungle there is a *fully collapsed jungle* representing the same terms most efficiently: multiple occurrences of (sub-)terms are represented only once. This fully collapsed jungle is uniquely determined up to isomorphism and can be generated from the given jungle by application of *folding rules*.

2.1 General Assumption

Let $SIG = (S, OP)$ be an arbitrary, but fixed signature. All hypergraphs considered in the following are hypergraphs over SIG, i.e., nodes are labeled with sorts from S and hyperedges are labeled with 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 $e \in E_G$, $m_G(e) = op : s_1 \dots s_k \rightarrow s$ implies $l^*_G(s_G(e)) = s$ and $l^*_G(t_G(e)) = s_1 \dots s_k$.

Remarks

1. Each hyperedge in a jungle labeled with an operation symbol $op : s_1 \dots s_k \rightarrow s$ has a single source node labeled with s and a sequence of k (not necessarily distinct) target nodes labelled with $s_1, \dots, s_k$. The sequence of target nodes is empty if op is a constant symbol.

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

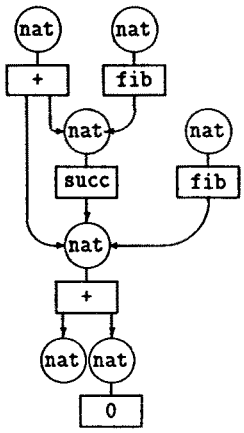
2.3 Example

Let SIG contain a sort `nat` and operation symbols $0 : \rightarrow$ `nat`, `succ`, `fib` : `nat` $\rightarrow$ `nat`, $+$: `nat nat` $\rightarrow$ `nat`. Then the following hypergraph is a jungle over SIG.

---

[1] A hypergraph is said to be acyclic if its underlying bipartite graph is acyclic.

[2] The outdegree of a node v in a hypergraph G is defined to be the outdegree of v in the underlying bipartite graph. I.e., $outdegree_G(v) = \sum_{e \in E_G} \#(v, s_G(e))$ where $\#(v, s_G(e))$ denotes the number of occurrences of v in the string $s_G(e)$.



Nodes are drawn as circles and hyperedges as boxes, both with their labels written inside. A line without arrow-head connects a hyperdege with its unique source node, while arrows point to the target nodes (if there are any). The arrows are arranged from left to right in the order given by the target mapping. □

In addition to the graph-theoretic definition 2.2, the set of all jungles can be characterized by a set of generating hypergraph rules: each jungle can be constructed from the empty hypergraph by application of these rules. The interested reader may consult [HKP 88] where this characterization is used to establish a structural induction principle and an efficient parsing algorithm for jungles. Here we are not concerned with these topics but proceed with the relationship between jungles and terms.

Let us consider a jungle G where each node has an outgoing hyperedge. Then for each node v in G we can construct a term $term_G(v)$ by descending along the hyperedges and collecting the hyperedge labels, starting with the unique hyperedge outgoing from v. This procedure terminates since G is acyclic and yields a unique term since all nodes in G have outdegree one. Vice versa, given a variable-free term t it is straightforward to construct a jungle representing t (e.g. by using the well-known tree representation for terms).

However, since we want to translate term rewrite rules into jungle rewrite rules we also have to represent variables in jungles. The problem is that variables cannot be represented as labels because the application of hypergraph rules is based on hypergraph morphisms which preserve labels. Moreover, in the case of multiple occurrences of a variable x in the left-hand side of a jungle rewrite rule we would have to make sure that all instantiations of x are equal before we apply the rule.

Our solution is to use nodes without outgoing hyperedges as variables. Such a variable x can be mapped to any jungle node v by a hypergraph morphism provided that x and v are labeled with the same sort. Since variables are nodes rather than labels, each variable in a jungle occurs only once. (The consequences for the handling of non-left-linear term rewrite rules are discussed in chapter 3.)

2.4 Definition (Variables of a Jungle)
Let G be a jungle. Then $VAR_G = \{v \in V_G \mid outdegree_G(v) = 0\}$ is the set of all *variables* in G. □

2.5 Lemma (Terms Represented by a Jungle)
Let G be a jungle. Then

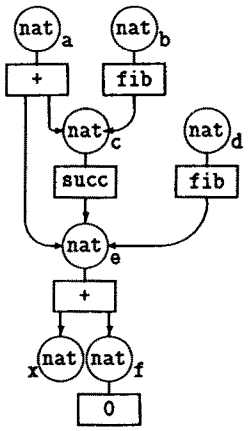
$$term_G(v) = \begin{cases} v & \text{if } v \in VAR_G, \\ m_G(e) & \text{if there is } e \in E_G \text{ with } s_G(e) = v \text{ and } t_G(e) = \lambda, \\ m_G(e)(term_G(v_1), \ldots, term_G(v_n)) & \text{if there is } e \in E_G \text{ with } s_G(e) = v \text{ and } t_G(e) = v_1 \ldots v_n \end{cases}$$

defines a function $term_G : V_G \to T_{SIG}(VAR_G)$. [3] □

Notation
The set $term_G(V_G)$ of all terms represented by a jungle G is denoted by $TERM_G$. □

[3] Given a variable set X, $T_{SIG}(X)$ denotes the set of all terms over X.

2.6 Example
Below we show the jungle G of example 2.3 together with its node names and list the terms represented by G. (We write terms with binary operators in infix notation.)



$$
\begin{aligned}
term_G(\mathtt{a}) &= \mathtt{x}+0+\mathtt{succ}(\mathtt{x}+0)\\
term_G(\mathtt{b}) &= \mathtt{fib}(\mathtt{succ}(\mathtt{x}+0))\\
term_G(\mathtt{c}) &= \mathtt{succ}(\mathtt{x}+0)\\
term_G(\mathtt{d}) &= \mathtt{fib}(\mathtt{x}+0)\\
term_G(\mathtt{e}) &= \mathtt{x}+0\\
term_G(\mathtt{f}) &= 0\\
term_G(\mathtt{x}) &= \mathtt{x}
\end{aligned}
$$

□

When comparing two different jungles it is natural to ask whether these jungles represent the same terms. The following definition provides the set of all jungles with an equivalence relation.

2.7 Definition (Equivalence)
Two jungles G and H are called *equivalent* if $TERM_G = TERM_H$. □

To represent a set T of terms by a jungle one may choose between equivalent jungles which represent T more or less redundantly. In fact each equivalence class of jungles representing a set of terms with at least one nonvariable term contains arbitrary large jungles. However, each equivalence class contains smallest jungles in which each two different nodes represent different terms.

2.8 Definition (Fully Collapsed Jungle)
A jungle G is said to be *fully collapsed* if for all $v_1, v_2 \in V_G$, $term_G(v_1) = term_G(v_2)$ implies $v_1 = v_2$ (i.e., if $term_G$ is an injective function). □

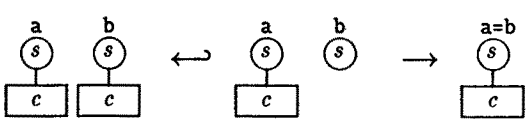
It turns out that fully collapsed jungles are, up to isomorphism, unique representatives of their equivalence classes.

2.9 Uniqueness Theorem
Equivalent fully collapsed jungles are isomorphic. □

Given a jungle, we want to generate the equivalent fully collapsed jungle by application of special hypergraph rules, called folding rules. To explain the idea behind these rules, consider a jungle G and two hyperedges e_1, e_2 in G which have the same labels and target nodes but different sources. Then $s_G(e_1)$ and $s_G(e_2)$ represent the same term. Identifying both source nodes and both edges yields a jungle equivalent to G.

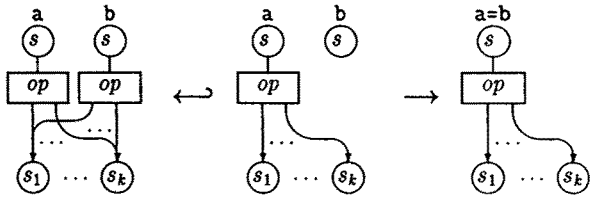
2.10 Definition (Folding)
The set FO of *folding rules* contains the following hypergraph rules: [4]

For each constant symbol $c :\rightarrow s$:



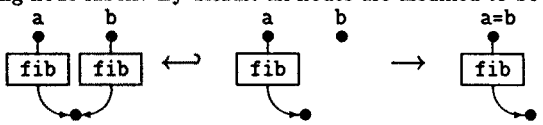
[4] The notation a=b expresses that the nodes a and b are identified by the right-hand morphism.

For each operation symbol $op : s_1 \dots s_k \to s$:



Let G, H be jungles. Then a direct derivation $G \underset{FO}{\Longrightarrow} H$ is called a *folding step* and a derivation $G \overset{*}{\underset{FO}{\Longrightarrow}} H$ is called a *folding*. □

2.11 Example (Folding)
Below we show the folding rule for the operation symbol `fib`. (Here and in the following examples we desist from indicating node labels. By default all nodes are assumed to be labeled with "`nat`").



An application of this rule is shown in figure 2. The left and right occurrences of the folding rule are drawn with dashed hyperedges and unfilled nodes. □

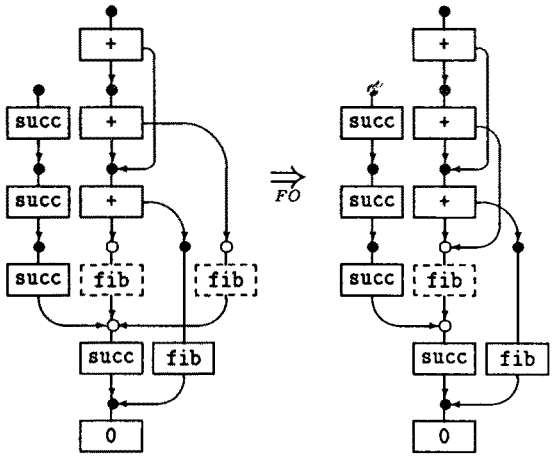


Figure 2: A folding step

By the definition of folding rules it is clear that applying a folding rule to a jungle always yields a jungle and, since each folding step deletes an edge, that each sequence of folding steps terminates. Moreover, given jungles G, H_1, H_2, and folding steps $H_1 \underset{FO}{\Longleftarrow} G \underset{FO}{\Longrightarrow} H_2$ there always is a jungle X such that $H_1 \overset{\lambda}{\underset{FO}{\Longrightarrow}} X \overset{\lambda}{\underset{FO}{\Longleftarrow}} H_2$. [5]

2.12 Theorem (Properties of Folding Steps)
1. $\underset{FO}{\Longrightarrow}$ preserves jungles and is terminating and strongly confluent.
2. A jungle G is fully collapsed if and only if there is no folding rule applicable to G. □

Given a jungle G, we know from the above theorem that the algorithm *apply folding rules as long as possible* terminates with a fully collapsed jungle H. Since folding preserves terms, [6] H is the unique fully collapsed jungle equivalent to G.

[5] Given hypergraphs G and H, we write $G \overset{\lambda}{\Rightarrow} H$ if $G \Rightarrow H$ or $G \cong H$.

[6] Here we already assume that folding steps do not rename variables. This is stated explicitly in assumption 3.6.

3 Jungle Evaluation

3.1 General Assumption

For the rest of this paper, TR denotes an arbitrary, but fixed term rewriting system[7] and $\rightarrow$ its rewrite relation. □

We want to construct a hypergraph rule $p = (L \leftarrow K \rightarrow R)$ that, when applied to some jungle G, performs rewriting according to a given term rewrite rule $l \rightarrow r$. How must p be defined?

The left-hand side L shall represent l in such a way that whenever there is a jungle L' representing l, there is a hypergraph morphism $L \rightarrow L'$ (since then p is applicable to each jungle containing L' as a subhypergraph). Therefore L has to be as little collapsed as possible.

The gluing hypergraph K contains those parts of L which are preserved by the application of p. We want to preserve all arguments of the topmost operation in l since these terms may be also the arguments of other operation symbols in G.

Finally, the right-hand side R has to represent r as well as the preserved subterms of l. We choose R to be fully collapsed since then R is the smallest jungle meeting this requirement.

3.2 Definition (Evaluation Rule and Evaluation Step)

Let $l \rightarrow r$ be a rewrite rule in TR. A hypergraph rule $(L \hookleftarrow K \xrightarrow{b} R)$[8] is called an *evaluation rule for* $l \rightarrow r$ if the following conditions are satisfied:

1. L is a variable-collapsed tree[9] with $term_L(root_L) = l$.
2. K is the subhypergraph of L obtained by removing the edge with source $root_L$.
3. R is a fully collapsed jungle such that $TERM_R$ contains all subterms of r and all proper subterms of l. [10]
4. For each $v \in V_K$, $term_R(b_V(v)) = \begin{cases} r & \text{if } v = root_L, \\ term_K(v) & \text{otherwise .} \end{cases}$

For each jungle G, a direct derivation $G \Rightarrow H$ through an evaluation rule is called an *evaluation step*. □

The application of evaluation rules to jungles always yields jungles again.

3.3 Theorem (Evaluation Steps Preserve Jungles)

Let G be a jungle and $G \Rightarrow H$ be an evaluation step. Then H is a jungle, too. □

3.4 Example (Jungle Evaluation)

Figure 3 below shows an evaluation rule for the recursive rewrite rule of `fib` given in the introduction.

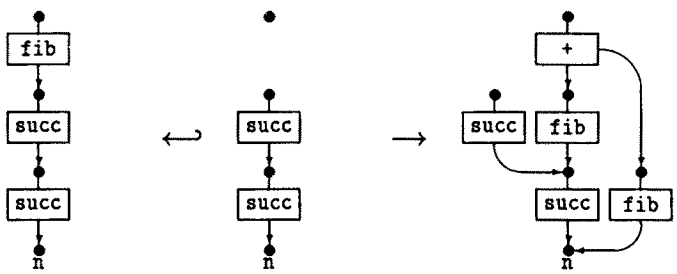


Figure 3: Evaluation rule for `fib(succ(succ(n)))→fib(succ(n))+fib(n)`

In figure 4 the application of this evaluation rule to a jungle is depicted. (Figure 1 in the introduction shows two applications of the same rule.) The term at the root `b` of the occurrence of the rule is

[7] I.e., each rewrite rule $l \rightarrow r$ in TR consists of two terms l and r of equal sort such that l is not a variable and all variables in r occur already in l.

[8] "$L \hookleftarrow K$" denotes the inclusion of K into L.

[9] I.e., there is a unique node $root_L$ in L with $indegree_L(root_L) = 0$ and for each node v with $indegree_L(v) > 1$ we have $v \in VAR_L$.

[10] Where a subterm t of l is called *proper* if $t \neq l$.

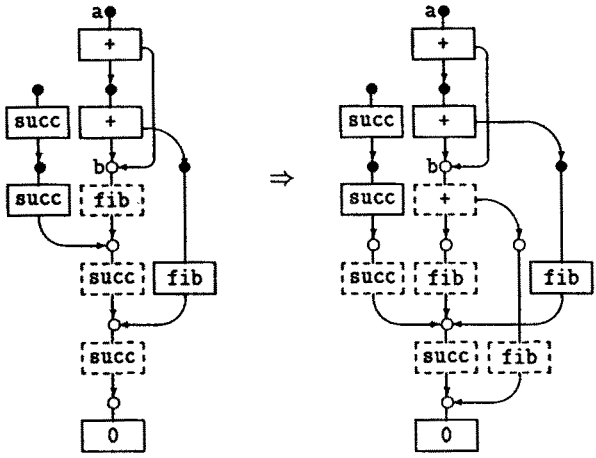


Figure 4: Application of the evaluation rule of figure 3 to a jungle

rewritten according to the underlying term rewrite rule. Terms at nodes which refer to b are rewritten as often as they refer to b (e.g., the term at a is rewritten twice). Terms at all other nodes are not changed. □

To be able to formally describe the effect of an evaluation step $G \Rightarrow H$ on the terms represented by G we define a node mapping from V_G to V_H.

3.5 Definition (Track)

1. Let $d : G \Rightarrow H$ be an evaluation step or a folding step, given by the diagram

$$\begin{array}{ccccc} L & \xleftarrow{b_1} & K & \xrightarrow{b_2} & R \\ \downarrow & & \downarrow & & \downarrow \\ G & \xleftarrow{c_1} & D & \xrightarrow{c_2} & H \end{array}$$

Since $b_{1,V}$ is bijective, pushout properties imply that $c_{1,V}$ is bijective, too. We define $track_d : V_G \to V_H$ by $track_d = c_{2,V} \circ c_{1,V}^{-1}$ where $c_{1,V}^{-1} : V_G \to V_D$ is the inverse of $c_{1,V}$.

2. Let $d : G \cong G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n = H$ be a derivation given by an isomorphism $f : G \to G_0$ and evaluation or folding steps $d_i : G_i \Rightarrow G_{i+1}$ for $i = 0, \ldots, n-1$. Then $track_d$ is defined by $track_d = track_{d_{n-1}} \circ \ldots \circ track_{d_0} \circ f_V$. □

For each hypergraph derivation $G \overset{*}{\Rightarrow} H$ the hypergraph H is uniquely determined only up to isomorphism. In particular, the variables of a jungle may be renamed by derivations. To avoid a cumbersome handling with variable renamings we want to assume that derivations over evaluation and folding rules do not rename the variables in a jungle. This is not a severe restriction since for each such derivation $G \overset{*}{\Rightarrow} H$ one can rename the nodes of H to obtain an isomorphic jungle H' such that $G \overset{*}{\Rightarrow} H'$ satisfies the following assumption.

3.6 General Assumption

Let $d : G \overset{*}{\Rightarrow} H$ be a derivation over evaluation and folding rules. Then we assume $track_d(x) = x$ for each variable x in G. □

By the following soundness theorem, evaluation steps perform term rewriting on the terms represented by a jungle. In general, a single evaluation step $G \underset{p}{\Rightarrow} H$ corresponds to multiple applications of the rewrite rule underlying p to several terms in $TERM_G$.

3.7 Theorem (Soundness of Evaluation Steps)

Let p be an evaluation rule and $d : G \underset{p}{\Rightarrow} H$ be an evaluation step. Then for each $v \in V_G$,

$$term_G(v) \xrightarrow{n} term_H(track_d(v))^{11}$$

[11] $\xrightarrow{n}$ denotes the n-fold composition of $\to$.

where n is the number of paths[12] from v to the root of the occurrence of p in G. □

In the following we want to manipulate jungles by application of both evaluation rules and folding rules.

3.8 Definition (Jungle Reduction)
Let $d : G \Rightarrow H$ be an evaluation or a folding step. Then d is called a *reduction step*. A derivation consisting of reduction steps is said to be a *reduction*.

A jungle G is called *reduced* if there is no reduction step $G \Rightarrow H$. □

3.9 Theorem (Normalization of Reduction)
A jungle G is reduced if and only if G is fully collapsed and all terms in $TERM_G$ are normal forms[13]. □

We want to emphasize that folding is necessary for the above result. More precisely, the statement *a jungle G is evaluated if and only if all terms in $TERM_G$ are normal forms* does not hold as long as TR contains non-left-linear rewrite rules. As an example, consider the rewrite rule `f(x,x)` → `c` and a corresponding evaluation rule p. The rewrite rule is applicable to the term `f(c,c)` while p can be applied to a jungle representing `f(c,c)` only if both occurrences of `c` are represented by the same node. The reason is that both occurrences of the variable `x` are represented by a single node in the left-hand side of p.

It should be noted that in general jungle reduction cannot compute all normal forms of a term: Let `h` be an operation symbol and `c` → `d` and `c` → `e` be rewrite rules with constant symbols `c`, `d`, and `e`. Then the fully collapsed jungle representing `h(c,c)` can only be reduced to jungles representing `h(d,d)` or `h(e,e)` although `h(d,e)` and `h(e,d)` are further normal forms of `h(c,c)`.

In the following we look for conditions under which jungle reduction is terminating and confluent[14] It turns out that termination of term rewriting carries over to jungle reduction without restriction.

3.10 Termination Theorem
If term rewriting is terminating, then jungle reduction is terminating, too. □

It would be nice if confluence would carry over from term rewriting to jungle reduction as smoothly as termination does. However, this is not the case, as the following example shows.

Let `c` → `f(d)`, `c` → `f(e)`, and `f(d)` → `f(e)` be rewrite rules for an operation symbol `f` and constant symbols `c`, `d`, and `e`. Figure 5 demonstrates that term rewriting is confluent, whereas jungle reduction yields distinct normal forms. This is because the argument `d` is removed when applying the rewrite

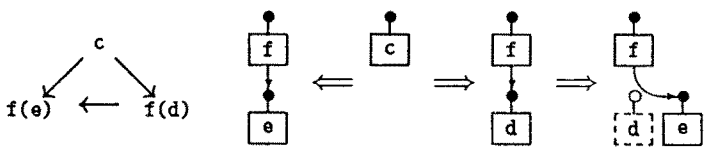


Figure 5: Non-confluence caused by garbage

rule `f(d)` → `f(e)`, while the corresponding subjungle (drawn with dashed box and unfilled circle) is preserved by the reduction. This subjungle is *garbage* with respect to the term rewriting performed.

In the following, we distinguish certain nodes in a jungle as "points of interest" and consider reductions that keep track of these points.

3.11 Definition (Pointed Jungle and Essential Jungle)
1. A *pointed jungle* is a pair (G, P_G) where G is a jungle and $P_G \subseteq V_G$. We may denote a pointed jungle only by its jungle component.

2. Let G be a pointed jungle. Then the *essential jungle* $G^\bullet$ is the subjungle of G consisting of all nodes and edges that are reachable from the nodes in P_G. □

Given a pointed jungle G, we consider the nodes and edges in $G - G^\bullet$ as *garbage*. It is possible to remove garbage from jungles explicitely by *garbage collecting rules* (which are the inverses of the jungle

[12] A *path* in a hypergraph is a path in the underlying bipartite graph.

[13] A term t is a *normal form* if no rewrite rule in TR is applicable to t.

[14] Let A be a set and $\rightarrow$ be a binary relation on A. $\rightarrow$ is called *terminating* if there is no infinite chain $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \ldots$ We say $\rightarrow$ is *confluent* if for all $a, a_1, a_2 \in A$, $a_1 \overset{*}{\leftarrow} a \overset{*}{\rightarrow} a_2$ implies that there is $a' \in A$ such that $a_1 \overset{*}{\rightarrow} a' \overset{*}{\leftarrow} a_2$.

generating rules defined in [HKP 88]). Here we just require that reductions preserve the distinguished nodes.

3.12 Definition (Pointed Reduction)
Let (G, P_G) and (H, P_H) be pointed jungles. Then a reduction $d : G \stackrel{*}{\Rightarrow} H$ is said to be *pointed* if $P_H = track_d(P_G)$. We write $G \stackrel{*}{\Rightarrow}\bullet H$ if d is pointed. □

With the notions just introduced, we are able to state that reduction is confluent "up to garbage" if term rewriting is terminating and confluent. Termination is needed because reduction is not confluent for some non-terminating but confluent term rewriting systems (cf. [HP 88] for an example).

3.13 Confluence Theorem
Let $\rightarrow$ be terminating and confluent and let $H_1 \bullet\!\stackrel{*}{\Leftarrow} G \stackrel{*}{\Rightarrow}\bullet H_2$ be pointed reductions. Then there are pointed reductions $H_1 \stackrel{*}{\Rightarrow}\bullet X_1$ and $H_2 \stackrel{*}{\Rightarrow}\bullet X_2$ such that $X_1^\bullet = X_2^\bullet$. □

4 Discussion and Outlook

The results of the previous sections can be used to show the correctness of the following procedure for computing normal forms of arbitrary terminating and confluent term rewriting systems.

4.1 Computing Term Normal Forms by Jungle Evaluation
Let TR be a terminating and confluent term rewriting system.

In order to compute the normal form of some term t, proceed as follows:

1. Choose a pointed jungle $(G, \{p\})$ with $term_G(p) = t$.
2. Construct a pointed reduction $d : G \stackrel{*}{\Rightarrow}\bullet H$ by applying evaluation and folding rules as long as possible.
3. Deliver $u = term_H(track_d(p))$ as a result.

By the termination theorem 3.10, step 2 terminates with H reduced, so u is a normal form according to the normalization theorem 3.9. Moreover, the soundness theorem 3.7 yields $t \stackrel{*}{\rightarrow} u$. Thus u is the unique normal form of t since $\rightarrow$ is confluent. □

4.2 Example (Efficiency of Evaluating Fibonacci Numbers)
Consider again the term rewrite rules for computing Fibonacci numbers shown in example 1.1.

We compare the efficiency of term rewriting and jungle reduction in terms of the steps needed to obtain normal forms. The Fibonacci number of 4 is computed in 9 term rewriting steps:

$$\begin{aligned}
\mathtt{fib}(\mathtt{succ}^4(0)) &\rightarrow \mathtt{fib}(\mathtt{succ}^3(0)) + \mathtt{fib}(\mathtt{succ}^2(0)) \\
&\rightarrow \mathtt{fib}(\mathtt{succ}^2(0)) + \mathtt{fib}(\mathtt{succ}(0)) + \mathtt{fib}(\mathtt{succ}^2(0)) \\
&\rightarrow \mathtt{fib}(\mathtt{succ}(0)) + \mathtt{fib}(0) + \mathtt{fib}(\mathtt{succ}(0)) + \mathtt{fib}(\mathtt{succ}^2(0)) \\
&\rightarrow \mathtt{fib}(\mathtt{succ}(0)) + \mathtt{fib}(0) + \mathtt{fib}(\mathtt{succ}(0)) + \mathtt{fib}(\mathtt{succ}(0)) + \mathtt{fib}(0) \\
&\stackrel{5}{\rightarrow} \mathtt{succ}(0) + 0 + \mathtt{succ}(0) + \mathtt{succ}(0) + 0
\end{aligned}$$

In general the number of steps for rewriting a term $\mathtt{fib}(\mathtt{succ}^n(0))$ grows exponentially with n, more precisely with ϕ^n, where $\phi = (1+\sqrt{5})/2 \approx 1.6180$ is the golden ratio. The size of the terms also grows exponentially.

A jungle reduction sequence corresponding to the above rewrite sequence is used as a running example throughout this paper. Figure 1 shows the first two evaluation steps and the first folding, while figure 4 shows the third evaluation step. Figure 2 shows the second folding step. Two applications of the non-recursive evaluation rules (which are not shown here) finish the reduction. Thus, only 5 evaluation steps and 2 folding steps are needed.

If, as in this example, the recursive fib-rule is always applied to the biggest argument and folding is done as early as possible, computing $\mathtt{fib}(\mathtt{succ}^n(0))$ requires only $n+1$ evaluation steps and $n-1$ folding steps, i.e. the number of reduction steps grows only linearly with n. □

4.3 Outlook
The work presented here should be continued in various directions:

The procedure described in 4.1 is still rather general in that reduction steps are performed in a completely arbitrary order. This freedom can be used to develop strategies for constructing optimal reductions.

We can consider *parallel jungle reduction*. Since we have used a standard type of graph grammars where parallelism has been studied, it should be rather easy to proceed in that direction.

Also, we would like to investigate jungle rules which *cannot* be modelled by term rewriting, e.g. jungle rules with more than one root on their left-hand sides. The folding rules are a first step into that direction. Such rules can perform simultaneous replacement of several patterns which share variables.

Furthermore, we would like to obtain precise measures for the complexity of jungle reduction to compare it with the complexity of term rewriting. In particular, we are interested to find out how efficient folding rules can be implemented and how much folding gains in practical implementations.

Finally, we are confident that our approach can be used to model the interpretation of functional programming languages adequately. It seems as if graph rewriting is one of the key concepts for developing efficient interpreters and compilers for functional languages (cf. [Pey 87]).

5 Appendix: Hypergraphs and Hypergraph Derivations

This appendix summarizes all notions concerning hypergraphs and hypergraph derivations that are needed for our purposes.

We adopt the hypergraph definition given in [HKP 88]. However, we consider a kind of hypergraph replacement which corresponds to the general case of graph replacement in the algebraic approach of graph grammars (see [Ehr 79]). The notions of graph grammar theory (which can be found, e.g., in [Ehr 79] and [Kre 87]) are adapted to the hypergraph case.

5.1 Hypergraphs

1. Let $C = (C_V, C_E)$ be a pair of sets, called *node* resp. *edge labels*.

2. A *hypergraph* $G = (V_G, E_G, s_G, t_G, l_G, m_G)$ over C consists of a finite set V_G of *nodes*, a finite set E_G of *hyperedges* (or *edges* for short), two mappings $s_G : E_G \to V_G^*$ and $t_G : E_G \to V_G^*$, assigning a string of *source nodes* and a string of *target nodes* to each hyperedge, and two mappings $l_G : V_G \to C_V$ and $m_G : E_G \to C_E$, assigning labels to nodes and hyperedges.

3. Given an edge e in a hypergraph G, $NODES_G(e)$ denotes the set of all nodes occurring in $s_G(e)$ and $t_G(e)$.

4. Let G, H be hypergraphs. 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 of H.

Remark

Each hypergraph corresponds to a bipartite graph and vice versa: To transform a hypergraph into a bipartite graph (called the *underlying bipartite graph*), the hypergraph nodes become graph nodes of the one sort and the hyperedges become nodes of the other sort; for each hyperedge e with source string $s(e)$ and target string $t(e)$ one constructs $|s(e)| + |t(e)|$ edges. Conversely, given a bipartite graph, nodes of one sort are taken as hypergraph nodes and nodes of the other sort as hyperedges.

5.2 Hypergraph Morphisms

1. Let G, H be hypergraphs. A pair of mappings $f = (f_V : V_G \to V_H,$
$f_E : E_G \to E_H)$ is a *hypergraph morphism* from G to H, denoted by $f : G \to H$, if $s_H \circ f_E = f_V^* \circ s_G$, $t_H \circ f_E = f_V^* \circ t_G$, $l_H \circ f_V = l_G$, and $m_H \circ f_E = m_G$. f is called *injective* (*surjective*, *bijective*) if both f_V and f_E are injective (surjective, bijective).

2. A bijective hypergraph morphism is called an *isomorphism*. Two hypergraphs G and H are *isomorphic*, denoted by $G \cong H$, if there exists an isomorphism from G to H.

3. Let $f : G \to H$ be a hypergraph morphism and $U \subseteq G$. Then $f_V(V_U)$ and $f_E(E_U)$ determine a subhypergraph of H which is denoted by $f(U)$.

5.3 Hypergraph Pushouts

Let $b : K \to B$ and $d : K \to D$ be two hypergraph morphisms. Then a hypergraph H together with two hypergraph morphisms $h : B \to H$ and $c : D \to H$ is called a *pushout* (or *gluing*) *of b and d* if the following conditions are satisfied:

Commutativity

$h \circ b = c \circ d$.

Universal Property

For all hypergraphs H' and hypergraph morphisms $h' : B \to H'$ and $c' : D \to H'$ with

$h' \circ b = c' \circ d$ there exists a unique hypergraph morphism $f : H \to H'$ such that $f \circ h = h'$ and $f \circ c = c'$.

If H together with h and c is a pushout of b and d we also call the following diagram a pushout:

$$\begin{array}{ccc} K & \xrightarrow{b} & B \\ {\scriptstyle d}\downarrow & & \downarrow{\scriptstyle h} \\ D & \xrightarrow{c} & H \end{array}$$

Remarks

1. The hypergraph H is determined uniquely up to isomorphism by the pushout properties.
2. For the construction of hypergraph pushouts we refer to [HP 88] (see also [Ehr 79] for the graph case).

5.4 Hypergraph Rules

A *hypergraph rule* p is a pair of hypergraph morphisms of the form $p = (L \leftarrow K \rightarrow R)$. L, R, and K are called the *left-hand side*, the *right-hand side*, and the *gluing hypergraph* of p, respectively.

5.5 Hypergraph Derivations

1. Let G, H be hypergraphs, $p = (L \leftarrow K \rightarrow R)$ be a hypergraph rule, and $g : L \to G$ be a hypergraph morphism. Then G *directly derives* H *through* p *and* g, denoted by $G \underset{p,g}{\Longrightarrow} H$, if there are two pushouts of the following form:

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

$g(L)$ and $h(R)$ are called the *left* resp. *right occurrence* of p.

2. Let G, H be hypergraphs and P be a set of hypergraph rules. Then a *derivation from* G *to* H *over* P, denoted by $G \overset{*}{\underset{P}{\Longrightarrow}} H$, is a sequence of direct derivations of the form $G \cong G_0 \underset{p_1}{\Rightarrow} G_1 \underset{p_2}{\Rightarrow} \ldots \underset{p_n}{\Rightarrow} G_n = H$ where $p_1, \ldots, p_n \in P$.

5.6 Gluing Condition

Let G be a hypergraph and $p = (L \xleftarrow{b_1} K \xrightarrow{b_2} R)$ be a hypergraph rule. Given a hypergraph morphism $g : L \to G$ there exists a pushout

$$\begin{array}{ccc} L & \xleftarrow{b_1} & K \\ {\scriptstyle g}\downarrow & (\star) & \downarrow \\ G & \leftarrow & D \end{array}$$

if and only if the *gluing condition* is satisfied which consists of the following two conditions:

Contact Condition

For all $e \in E_G - E_{g(L)}$: $NODES_G(e) \cap V_{g(L)} \subseteq V_{g(b_1(K))}$.

Identification Condition

For all $x, y \in L$[15] : $x \neq y$ and $g(x) = g(y)$ implies $x, y \in b_1(K)$.

Remark

In general there may be several nonisomorphic hypergraphs D such that $(\star)$ becomes a pushout. However, if b_1 is injective, then D is determined uniquely up to isomorphism.

5.7 Hypergraph Derivations

Let G be a hypergraph, $p = (L \xleftarrow{b_1} K \xrightarrow{b_2} R)$ be a hypergraph rule and $g : L \to G$ be a hypergraph morphism such that the gluing condition is satisfied. Then the following construction yields a direct derivation $G \underset{p,g}{\Rightarrow} H$.

(1) $D = G - (g(L) - g(b_1(K)))$ is a subhypergraph of G. [16]

(2) $D \to G$ is the inclusion of D into G.

(3) $d : K \to D$ is defined for each $x \in K$ by $d(x) = g(b_1(x))$.

(4) H together with $h : R \to H$ and $c : D \to H$ is the pushout of $b_2 : K \to R$ and $d : K \to D$.

---

[15] "$x, y \in L$" is a short notation for "$x, y \in V_L$ or $x, y \in E_L$" and is used when nodes and edges have not to be distinguished.

[16] More precisely, D is the subhypergraph with $V_D = V_G - (V_{g(L)} - V_{g(b_1(K))})$ and $E_D = E_G - (E_{g(L)} - E_{g(b_1(K))})$.

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 Rewriting.* Proc. PARLE, Lecture Notes in Comp. Sci. 259, 141-158 (1987)

[Der 85] N. Dershowitz: *Computing with Rewrite Systems.* Information and Control 64, 122-157 (1985)

[Ehr 79] H. Ehrig: *Introduction to the Algebraic Theory of Graph Grammars.* Proc. 1st Graph Grammar Workshop, Lecture Notes in Comp. Sci. 73, 1-69 (1979)

[EM 85] H. Ehrig, B. Mahr: *Fundamentals of Algebraic Specification 1 – Equations and Initial Semantics.* Springer, Monographs in Computer Science, New York–Berlin–Heidelberg (1985)

[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)

[FGJM 85] K. Futatsugi, J. Goguen, J.P. Jouannoud, J. Meseguer: *Principles of OBJ2.* Proc. 1985 Symposium on Principles of Programming Languages, 52-66 (1985)

[HKP 88] A. Habel, H.-J. Kreowski, D. Plump: *Jungle Evaluation.* To appear in Proc. Fifth Workshop on Specification of Abstract Data Types (1988)

[Hof 83] B. Hoffmann: *Compiler Generation: From Language Descriptions to Abstract Compilers.* Dissertation, TU Berlin (1983)

[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)

[HP 88] B. Hoffmann, D. Plump: *Jungle Evaluation for Efficient Term Rewriting.* Technical Report, Universität Bremen, (1988)

[Hsi 85] J. Hsiang: *Refutational Theorem Proving using Term Rewriting Systems.* Artificial Intelligence 25, 255-300 (1985)

[Klo 87] J.W. Klop: *Term Rewriting Systems: A Tutorial.* EATCS Bulletin 32, 143-182, (1987)

[Kre 87] H.-J. Kreowski: *Is Parallelism Already Concurrency? – Part 1: Derivations in Graph Grammars.* Proc. 3rd Graph Grammar Workshop, Lecture Notes in Comp. Sci. 291, 343-360 (1987)

[O'D 87] M.J. O'Donnell: *Term Rewriting Implementation of Equational Logic Programming.* Proc. 2nd Conference on Rewriting Techniques and Applications, Lecture Notes in Comp. Sci. 256, 1-12 (1987)

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

[Pad 83] P. Padawitz: *Correctness, Completeness, and Consistency of Equational Data Type Specifications.* Bericht Nr. 83-15, Technische Universität Berlin (1983)

[Pey 87] S.L. Peyton Jones: *The Implementation of Functional Programming Languages.* Prentice-Hall, Englewood Cliffs (1987)

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

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