Term Rewriting with Sharing and Memoïzation

Berthold Hoffmann, Universität Bremen*

Abstract

Jungle evaluation is an approach to define *term rewriting with sharing* based on graph grammars. This approach preserves important properties of term rewriting like termination, and confluence for terminating systems (under mild restrictions).

In this paper, term rewriting with sharing is further accelerated, by *memoïzation* known from functional programming languages: The result of evaluating a function with some arguments is *tabulated* so that it can be *looked up* later on when the function is re-applied to the same arguments.

We show that term rewriting with sharing and memoïzation is correct and complete w.r.t. jungle evaluation if the rules are non-overlapping and non-looping. Redundant re-evaluation of functions is avoided, independent of a particular strategy for applying evaluation rules.

1 Introduction

Term rewriting is a basis for prototyping algebraic specifications of abstract data types, and a foundation of functional programming languages (see [DJ90] and [Klo90] for overviews).

Basically, terms are *trees*, and rewriting is *subtree replacement*. This may be very expensive, both in time and space: the application of a rule may copy large subterms, and each copy has to be rewritten anew.

In the papers [HP88] and [HP91], D. Plump and the author have proposed *jungle evaluation* as an improved model for term rewriting *with sharing* where some sources of inefficiency are avoided:

- Terms are represented by acyclic (hyper-) graphs so that multiple occurrences of terms can be shared. These graphs are called *jungles*, a name coined in [HKP88].
- Rewriting is performed by graph replacement, specified by *evaluation rules* according to the algebraic theory of graph grammars (see, e.g., [Ehr79]). Evaluation rules make that terms are shared instead of being copied.

*Address: Fachbereich Mathematik und Informatik, Universität Bremen, Postfach 330 440, 2800 Bremen 33, Germany. Usenet: hof%Informatik.Uni-Bremen.DE *Published in the proceedings of the 3rd International Conference on Algebraic and Logic Programming (ALP'92), Springer LNCS 632, 128 – 142, 1992.*

- Additional *folding* graph rules can achieve maximal sharing of common subterms; so non-left-linear evaluation rules can be handled, and multiple evaluation of subterms can sometimes be avoided.

In this paper, we extend jungle evaluation by *memoïzation.* This is based on the wellknown optimization technique for functional programming languages: The result of a function application is *tabulated* so that it can be *looked up* later on if the function is re-applied to the same arguments, in order to avoid redundant re-evaluation.

We show that jungle evaluation with memoïzation is correct and complete w.r.t. jungle evaluation, for non-overlapping and non-looping rules.

Jungle evaluation with memoïzation is suited to define schemes for *full* and *lazy memoïzation* in the sense of [Hug85]. These schemes are independent of particular strategies for applying evaluation rules.

The paper is organized as follows: We start by defining (hyper-) graphs and jungles in section 2. In section 3 we recall the results of [HP88] and [HP91] which are relevant for this paper. In section 4 we define tabulating and lookup rules and show correctness and completeness. Full and lazy memoïzation schemes are defined in section 5. Finally, we summarize our results and point out possible directions of future research in section 6.

Due to the space restrictions, some less important proofs have been ommitted; they can be found in the technical report [Hof92].

Acknowledgements. This paper has been stimulated by Stefan Kahrs' efficient implementation of term rewriting languages using folding and full memoïzation (see [Kah92]). Detlef Plump has made valuable comments on an earlier version of this paper.

This work has been supported by the ESPRIT Basic Research Working Group 3299 CompuGraph, "Computing by Graph Transformation".

2 Jungles

Representing functional expressions with sharing requires graph-like structures. We use hypergraphs (over many-sorted signature), an extension of graphs where each hyperedge connects a sequence of source nodes to a sequence of target nodes.

2.1 Definition (Signature)
A *signature* is a system $\mathcal{Sig} = (\mathcal{S}, \mathcal{Op}, \mathrm{arg}, \mathrm{val})$ where $\mathcal{S}$ and $\mathcal{Op}$ are sets, and $\mathrm{arg} : \mathcal{Op} \to \mathcal{S}^*$ [1], $\mathrm{val} : \mathcal{Op} \to \mathcal{S}$ are functions. The elements of $\mathcal{S}$ and $\mathcal{Op}$ are called *sorts* and *operation symbols*, resp. We write $F : s_1 \dots s_k \to s \in \mathcal{Op}$ if we wish to indicate that an operation symbol $F \in \mathcal{Op}$ has $\mathrm{arg}(F) = s_1 \dots s_k$ and $\mathrm{val}(F) = s$.

For the rest of this paper, $\mathcal{Sig}$ denotes an arbitrary, fixed signature. □

2.2 Definition (Hypergraph, Hypergraph Morphism)
A *hypergraph (over $\mathcal{Sig}$)* is a system $H = (V, Y, s, t, \mathrm{sort}, \mathrm{op})$ where V is a set of *nodes*, Y

[1] A^* denotes the set of strings over a set A, including the empty string λ

is a set of *hyperedges*, $s, t : Y \to V^*$ are *source* and *target* mappings, and $\mathrm{sort} : V \to \mathcal{S}$ and $\mathrm{op} : Y \to \mathcal{O}p$ are mappings *labelling* nodes by sorts, and hyperedges by operation symbols.

Let G, H be hypergraphs.

A *(hypergraph) morphism* $f : G \to H$ is a pair of mappings $f = (f_V : V_G \to V_H, f_Y : Y_G \to Y_H)$ which preserve sources, targets, and labels, i.e., $s_H \circ f_Y = f_V^* \circ s_G$, $t_H \circ f_Y = f_V^* \circ t_G$ [2], $\mathrm{sort}_H \circ f_V = \mathrm{sort}_G$, and $\mathrm{op}_H \circ f_Y = \mathrm{op}_G$.

If both f_V and f_Y are bijective, f is called an *isomorphism*, and G and H are said to be *isomorphic*, written $G \cong H$.

Convention: For convenience, we mostly say *graph* instead of hypergraph, and *edge* instead of hyperedge. We write f for f_V or f_Y if no confusion arises. □

2.3 Definition (Dependency and Subgraph)
Let G be a graph with nodes $v, v' \in V_G$.

We say that v *depends on* v' if v occurs in the sources and v' in the targets of some edge $y \in Y_G$, and write $\prec_G$ for the transitive, and $\preceq_G$ for the transitive-reflexive closure of this relation.

$G\!\downarrow_v$ denotes the *(full) subgraph at* v, i.e. the graph with nodes $\overline{V} = \{\overline{v} \in V_G \mid v \preceq_G \overline{v}\}$ and edges $\{y \in Y_G \mid s_G(y)t_G(y) \in \overline{V}^*\}$. □

Not all graphs over $\mathcal{S}ig$ are suited to represent functional expressions: The sources and targets of edges must conform with the typing of $\mathcal{S}ig$, each node may only have at most one outgoing edge (in order to represent a unique expression), and no cycles may occur (in order to stay with finite expressions). Graphs meeting these requirements are called jungles, a name coined in [HKP88].

2.4 Definition (Jungle and Variable)
Let G be a graph over $\mathcal{S}ig$.

G is *compatibile (with $\mathcal{S}ig$)* if $\mathrm{sort}_G^* \circ t_G = \mathrm{arg} \circ \mathrm{op}_G$ and $\mathrm{sort}_G^* \circ s_G = \mathrm{val} \circ \mathrm{op}_G$. [3]

G is *univalent* if every node occurs at most once in the sources of at most one edge.

G is a *jungle* if it is compatibile, univalent, and acyclic (i.e. no node $v \in V_G$ satisfies $v \prec_G v$).

A node v in G is a *variable* if it does not appear in the sources of any edge of G. □

2.5 Example
Let $\mathcal{S}ig$ contain a sort nat and operation symbols $\mathsf{0} : \lambda \to \mathsf{nat}$, $\mathsf{S}, \mathsf{FIB} : \mathsf{nat} \to \mathsf{nat}$, $+ : \mathsf{nat}\ \mathsf{nat} \to \mathsf{nat}$. Then the graphs in figures 1 and 2 below are jungles over $\mathcal{S}ig$. □

Jungles can be distinguished by their degree of sharing.

2.6 Definition (Special Jungle)
A jungle G is called

[2] f^* denotes the homomorphic extension of a function $f : A \to B$ to strings A^*

[3] In the following, all graphs will be silently assumed to be compatible

1. *variable-collapsed* if only variables occur in the targets of more than one edge of G, or more than once in the target of a single edge of G.

2. *fully collapsed* if all isomorphic subgraphs are identical, i.e. if $G{\downarrow}_v \cong G{\downarrow}_{v'}$ implies $v = v'$ for all nodes $v, v' \in V_G$,.

A (variable-collapsed/fully collapsed) jungle G is called a *(variable-/fully) collapsed tree* if there is a unique root node, denoted $root_G$, which is not source of any edge, with $root_G \preceq_G v$ for all $v \in V_G$. □

3 Term Rewriting with Sharing

In this section we recall some results of [HP91] on term rewriting with sharing.

First we define the notions of hypergraph rule and hypergraph derivation used in this paper.

3.1 Definition (Contextual Hyperedge Replacement)

A *contextual hyperedge-replacement rule* (*CHR rule*, for short) is given by $p : L[y] \xrightarrow{b} R$, where L and R are hypergraphs, $y \in Y_L$ is a hyperedge, defining an *interface graph* $L\backslash y$ as L without y, and $b : L\backslash y \to R$ is a hypergraph morphism.

Let $p : L[y] \xrightarrow{b} R$ be a CHR rule.

A morphism $g : L \to G$ is an *occurrence of p in G* if it satisfies the *gluing condition*

$$\forall y' \in Y_L : g(y') = g(y) \text{ implies } y' = y \tag{1}$$

Let G and H be hypergraphs and $p : L[y] \xrightarrow{b} R$ be a CHR rule. Then G derives directly to H via p, written $G \underset{p}{\Longrightarrow} H$, if

- there is an occurrence $g : L \to G$ of p in G, and
- H is isomorphic to the hypergraph H' obtained from the disjoint union of $G\backslash g(y)$ and R by identifying all nodes and edges in $g(L\backslash y)$ with their counterparts in $b(L\backslash y)$.

We call $g(y)$ the *handle* of $G \underset{p}{\Longrightarrow} H$.

Let $\mathcal{P}$ and $\mathcal{Q}$ be sets of CHR rules. We write $G \underset{\mathcal{P}}{\Longrightarrow} H$ if $G \underset{p}{\Longrightarrow} H$ for some $p \in \mathcal{P}$, $G \overset{+}{\underset{\mathcal{P}}{\Longrightarrow}} H$ if there is a derivation sequence with $n > 0$ steps, and $G \overset{*}{\underset{\mathcal{P}}{\Longrightarrow}} H$ if $G \overset{+}{\underset{\mathcal{P}}{\Longrightarrow}} H$ or $G \cong H$. We abbreviate $\mathcal{P} \cup \mathcal{Q}$ by $\mathcal{P}\mathcal{Q}$, and $\underset{\mathcal{P}\cup\mathcal{Q}}{\Longrightarrow}$ by $\underset{\mathcal{P}\mathcal{Q}}{\Longrightarrow}$.

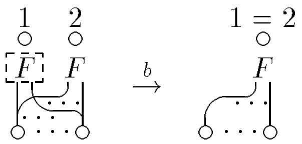
Each derivation $G \underset{\mathcal{P}}{\Longrightarrow} H$ induces a *track function* $tr : V_G \to V_H$ which is an isomorphism on V_G up to identifications in $b(L\backslash y)$, and can naturally be extended to derivation sequences by composition. □

CHR rules are a special kind of *hypergraph rules* according to the algebraic theory of graph grammars (double pushout approach). See [HP91] for details.

3.2 Definition (Evaluation and Folding Rules)
1. A CHR rule $p : L[y] \xrightarrow{b} R$ is an *evaluation rule* if

- L is a variable-collapsed tree with $root_L = t_L(y)$.
- R is obtained as a union of $L\backslash y$ with a collapsed tree N such that $root_L$ is identified with $root_N$, and each variable in N is identified with a variable in $L\backslash y$ other than $root_L$.
- b is the set inclusion with the exception that b identifies $root_L$ with $root_N$ if $N \subset L\backslash y$.

2. The folding rule $p : L[y] \xrightarrow{b} R$ for a function symbol $F : s_1 \ldots s_k \to s \in \mathcal{O}p$ (with $k \geq 0$) is defined by the picture below. (y is drawn in a dashed box; "1 = 2" in R indicates that b identifies the roots 1 and 2 of L.)



3. For the rest of this paper, we fix a set $\mathcal{E}$ of evaluation rules, and a set $\mathcal{F}$ of folding rules for $\mathcal{O}p$.

3.3 Example (Evaluation and Folding Rules and Steps)
In fig. 1 we show an evaluation rule and a folding rule.

In jungles, nodes are drawn as circles; we omit their labels because they can be deduced from the context. Edges are depicted by their labels which appear below their unique source nodes, with lines pointing to the target nodes (if there are any). The lines are arranged from left to right in the order given by the target mapping.

Fig. 2 shows an evaluation step and a folding step under the rules of figure 1. □

Jungle evaluation with folding is an adequate and efficient model for term rewriting. The reader is referred to [HP91] for details.

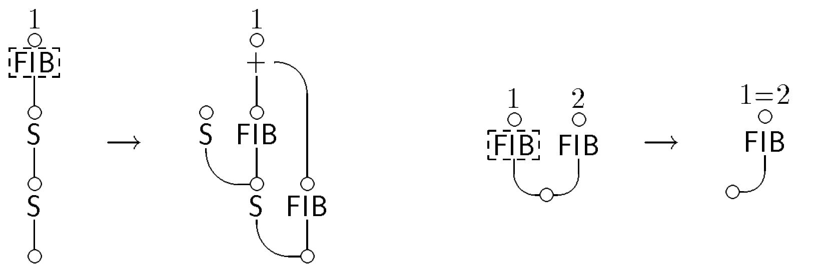


Figure 1: An evaluation rule and a folding rule (example 3.3)

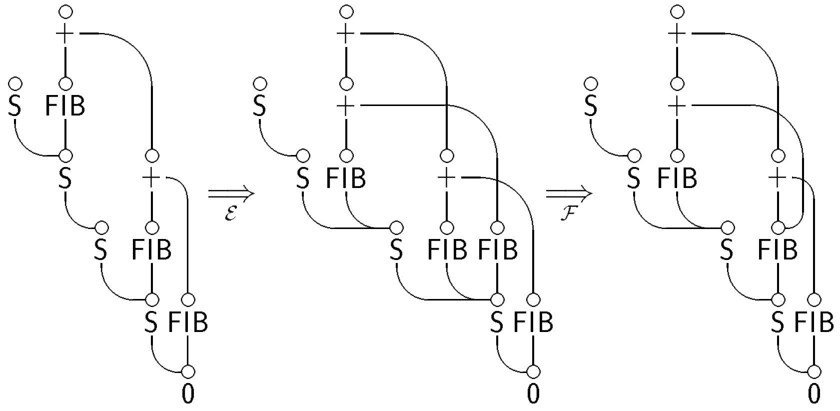


Figure 2: An evaluation and a folding step (example 3.3)

4 Tabulation and Lookup

In this section we define how the effect of an evaluation step is *tabulated* so that it can be *looked up* later on, in order to avoid re-evaluation. First we extend $\mathcal{Sig}$-graphs by edges which shall represent handles of previous evaluation steps.

4.1 Definition ($\mathcal{Sig}^\square$-Graph)
Let $\mathcal{Op}^\square$ be a set containing a *boxed function symbol* $\boxed{F} : s_1 \dots s_k \to s$ for each symbol $F : s_1 \dots s_k \to s \in \mathcal{Op}$, and consider a graph G over $\mathcal{Sig}^\square = (\mathcal{S}, \mathcal{Op} \cup \mathcal{Op}^\square)$.

G is a $\mathcal{Sig}^\square$*-graph* if it is compatible with $\mathcal{Sig}^\square$.

$\overline{G}$ denotes the $\mathcal{Op}$-labelled subgraph of G, and $Y_G^\square = Y_G - Y_{\overline{G}}$ denotes the set of boxed edges in G. $\square$

Jungle evaluation and folding rules need only slight modifications to realize tabulation and lookup of evaluation steps. Evaluation rules are modified so that the handle of the rule is not deleted, but replaced by a corresponding boxed edge which tabulates the result of the step. Folding rules remain unchanged. A new set of lookup rules folds an edge with an evaluated handle of the same label and targets.

4.2 Definition (Tabulating and Lookup Rules)
1. Let $p : L[y] \xrightarrow{b} R$ be an evaluation rule. The tabulating evaluation rule $p' : L[y] \xrightarrow{b} R'$ for p is obtained by extending R with an $\mathcal{Op}^\square$-edge y' such that $s_{R'}(y') = b^*(s_L(y))$, $t_{R'}(y') = b(t_L(y))$, and $\mathrm{op}_{R'}(y') = \boxed{F}$ if $\mathrm{op}_L(y) = F$.

2. Let $p : L[y] \xrightarrow{b} R$ be the folding rule of some function symbol $F \in \mathcal{Op}$, and let $y' \in Y_L$ with $y' \neq y$. The lookup rule $p' : L'[y] \xrightarrow{b} R'$ for p is obtained by re-labelling y' and $b(y')$ by $\boxed{F}$.

3. $\mathcal{T}$ denotes the set of tabulating evaluation rules for $\mathcal{E}$, and $\mathcal{L}$ the set of lookup rules for $\mathcal{Op}$. $\square$

4.3 Example (Tabulating and Lookup)
In fig. 3 we show the tabulating and lookup rule rule corresponding to example 3.3. Fig. 4 shows a tabulating and a folding step corresponding to example 3.3, followed by a lookup step.

In pictures of $\mathcal{S}ig^{\square}$-graphs we draw a thick arrow from boxed edges to their source nodes. $\square$

Tabulating rules are "robust" extensions of evaluation rules; tabulation and folding steps perform evaluation and folding steps on the underlying $\mathcal{S}ig$-graph of a $\mathcal{S}ig^{\square}$-graph.

4.4 Theorem (Correspondence)
For a $\mathcal{S}ig^{\square}$-graph G (with underlying $\mathcal{S}ig$-graph $\overline{G}$),

$$G \underset{\mathcal{TF}}{\Longrightarrow} H \text{ if and only if } \overline{G} \underset{\mathcal{EF}}{\Longrightarrow} H' \text{ with } H' \cong \overline{H}$$

Sketch of Proof: It is easy to show that $\overline{G} \underset{\mathcal{T}}{\Longrightarrow} H$ iff $\overline{G} \underset{\mathcal{E}}{\Longrightarrow} H'$ so that $H' \cong H\backslash\tilde{y}$ where $\tilde{y}$ is the track of the handle y in $\overline{G} \underset{\mathcal{T}}{\Longrightarrow} H$. Thus $\overline{G} \underset{\mathcal{TF}}{\Longrightarrow} H$ iff $\overline{G} \underset{\mathcal{EF}}{\Longrightarrow} H'$ with $H' \cong \overline{H}$.

Since the occurrences of tabulation, evaluation and folding steps are always in $\overline{G}$, the clip and join theorems in [Kre77] allow to extend this result to arbitrary $\mathcal{S}ig^{\square}$-graphs. $\square$

In the remainder of this section, we establish a correspondence of lookup steps with evaluation and folding steps which is similar to theorem 4.4: every lookup step should correspond to a (redundant) sequence of evaluation and folding steps.

However, such a correspondence cannot be expected to hold for arbitrarily constructed $\mathcal{S}ig^{\square}$-graphs. From now on, we therefore consider only $\mathcal{S}ig^{\square}$-graphs which have been derived from jungles by applying tabulating, folding and lookup rules, and call such graphs memo graphs. The boxed edges in such graphs shall record evaluation and folding steps. Memo graphs with this property are called *correct*.

4.5 Definition (Memo Graph, Handle Graph, Correct Memo Graph)
A $\mathcal{S}ig^{\square}$-graph G is a *memo graph* if there is a jungle G_0 such that $G_0 \overset{*}{\underset{\mathcal{TFL}}{\Longrightarrow}} G$.

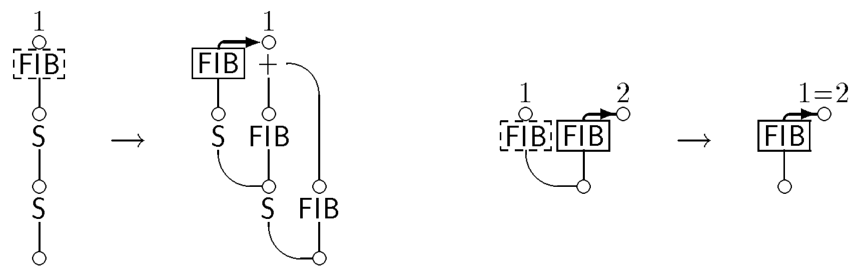


Figure 3: A tabulating and a lookup rule (example 4.3)

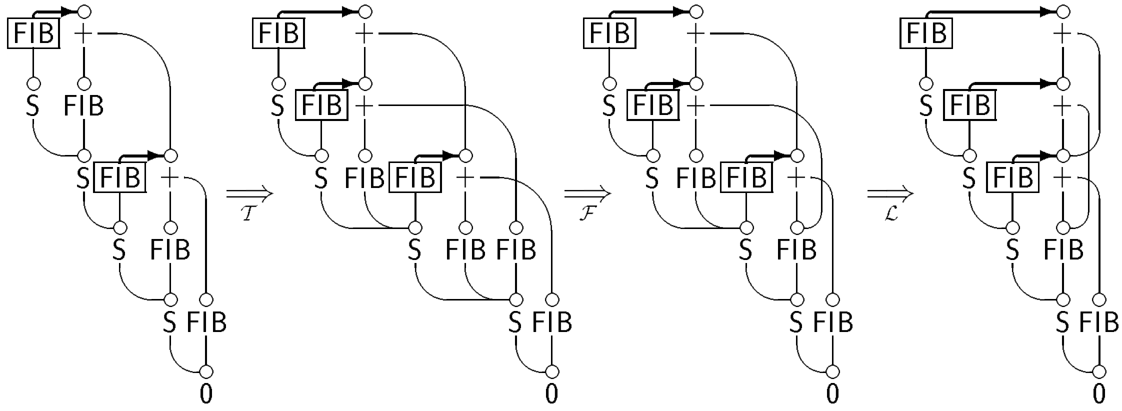


Figure 4: A tabulating, a folding and a lookup step (example 4.3)

Let $y \in Y_G^{\square}$ with target nodes $t_G(y) = v_1 \ldots v_k$ and $\mathrm{op}_G(y) = \boxed{F}$. The *handle graph* at y, denoted by $G{\downarrow_y}$, is obtained as the union of $\overline{G}{\downarrow_{v_1}} \cup \ldots \cup \overline{G}{\downarrow_{v_k}}$ with a new node v' with label $\mathrm{sort}_G(s_G(y))$ and a new edge y' with label F, source v', and target string $v_1 \ldots v_k$.

A memo graph G is *correct* (w.r.t. $\mathcal{EF}$) if all boxed edges $y \in Y_G^{\square}$ satisfy $G{\downarrow_y} \overset{+}{\underset{\mathcal{EF}}{\Longrightarrow}} G_y$ for some jungle G_y with $\overline{G}{\downarrow_{s_G(y)}} \subset G_y \subset \overline{G}$. □

Note: All memo graphs in figure 4 of example 4.3 are correct. □

4.6 Example (Cyclic Lookup)
Correct memo graphs are not closed under lookup steps. The tabulation and lookup steps in figure 5 introduce a cycle which cannot be produced by jungle evaluation and folding.

Note that the tabulation rule, and the corresponding evaluation rule, derives a graph wherein the start graph occurs again. Such evaluation rules are called looping, in analogy with a definition for term reawriting systems in [Der87], and have to be excluded in the following. □

4.7 Definition (Non-Looping)
Let $\mathcal{EF}$ be a set of evaluation and folding rules. $\mathcal{EF}$ is called *looping* if there is a derivation

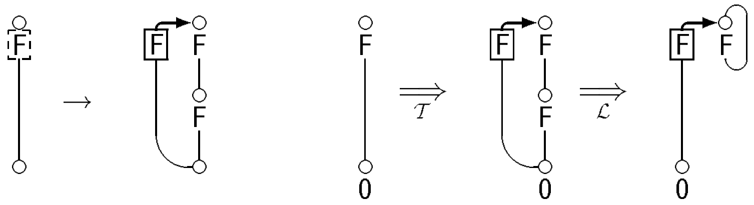


Figure 5: Looping tabulating rule and cyclic lookup step (example 4.6)

$G \overset{+}{\underset{\mathcal{EF}}{\Longrightarrow}} H$ containing at least one evaluation step so that there is a morphism $G \to H$; $\mathcal{EF}$ is called *non-looping* otherwise. □

Derivations by looping evaluation and folding rules are not terminating; the reverse is not true. It is undecidable whether a given set $\mathcal{EF}$ of evaluation and folding rules is looping or not (see [Der87, Pla85] for details).

4.8 Lemma (Preservation of Underlying Jungles)
Let G be a correct memo graph and $G \underset{\mathcal{TFL}}{\Longrightarrow} H$. Then $\overline{H}$ is a jungle if $\underset{\mathcal{EF}}{\Longrightarrow}$ is non-looping.

Proof: If $G \underset{\mathcal{TF}}{\Longrightarrow} H$, correspondence theorem 4.4 and jungle preservation shown in [HP91, lemma 4.5 and theorem 5.4] prove the claim.

Let therefore $p \in \mathcal{L}$ be the rule applied, $g : L \to G$ be the occurrence, and $\tilde{}$ the track of the derivation. Let y_1, y_2 be the edges in $g(Y_L)$, where y_1 is the handle of the step and consider the nodes $r_1 = s_G(y_1)$ and $r_2 = s_G(y_2)$ in G.

1. ($\overline{H}$ is univalent) Since $\tilde{r}_1 = \tilde{r}_2$, the outgoing edges of r_1 and r_2 are joined in $\tilde{r}_1$, up to the $\mathcal{O}p$-edge y_1 which is removed. Thus $\tilde{r}_1$ has again at most one outgoing $\mathcal{O}p$-edge. All other nodes in G preserve their outgoing edges so that $\overline{H}$ is univalent.

2. ($\overline{H}$ is acyclic) Any cycle in $\overline{H}$ is created by "gluing together" dependencies $r_1 \prec_{\overline{G}} r_2$ and $r_2 \prec_{\overline{G}} r_1$ since it would have already existed in $\overline{G}$ otherwise.

Correctness of G implies $G{\downarrow}_{y_2} \overset{+}{\underset{\mathcal{EF}}{\Longrightarrow}} G_y$ with $\overline{G}{\downarrow}_{r_2} \subset G_y$. If $r_2 \prec_{\overline{G}} r_1$ there is a morphism $G{\downarrow}_{r_1} \to G{\downarrow}_{r_2}$, because $G{\downarrow}_{r_1} \cong G{\downarrow}_{y_2}$ by definition of $\mathcal{L}$. This contradicts the assumption that $\mathcal{EF}$ is non-looping. □

4.9 Example (Incorrect Tabulating Evaluation)
The application of tabulating rules to a correct memo graph does not always yield a correct memo graph, see in figure 6.

Note that this happens because the occurrences of the tabulating and evaluation rules *overlap*, in the sense known from term rewriting, see [Klo91]. So we have to exclude this type of rules as well. □

4.10 Definition (Non-Overlapping)
A set $\mathcal{E}$ of evaluation rules is *non-overlapping* if for all derivations $G \underset{p_i}{\Longrightarrow} H_i$ via rules $p_i : L_i[y_i] \xrightarrow{b_i} R_i \in \mathcal{E}$ and occurrences $g_i : L_i \to G$ for $i = 1, 2$, $y_1 \in g_2(L_2)$ implies that $p_1 = p_2$ and $g_1(L_1) = g_2(L_2)$. □

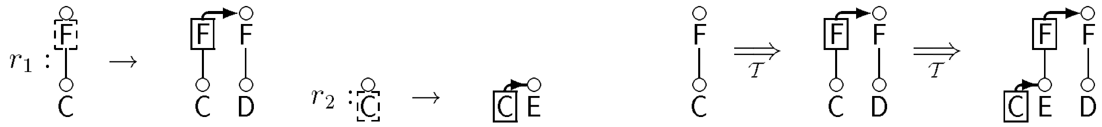


Figure 6: Overlapping rules and incorrect tabulation steps (example 4.9)

4.11 Theorem (Correctness)
Memo graphs for non-overlapping and non-looping $\mathcal{EF}$ are correct.

Proof: Jungles are trivial correct memo graphs since they contain no boxed edges at all.

We therefore assume $G \underset{\mathcal{TFL}}{\Longrightarrow} H$ for some correct memo graph G, and show that H is correct too.

Let $p \in \mathcal{TFL}$ be the rule, and $g : L \to G$ the occurrence used in the derivation. We write $\tilde{v}$, $\tilde{y}$, and $\tilde{G}_0$ for the tracks of arbitrary nodes $v \in V_G$, edges $y \in Y_G$, and subgraphs $G_0 \subset G$, resp.

We have to distinguish three cases.

1. If $p \in \mathcal{F}$, let y_i and r_i, $i = 1, 2$ be given as in the proof of lemma 4.8. Then $\overline{G}{\downarrow}_{r_1} \cong \overline{G}{\downarrow}_{r_2}$ and $\overline{H}{\downarrow}_{\tilde{r}_1} = \overline{H}{\downarrow}_{\tilde{r}_2}$ by the semantics of folding.

For all boxed edges $y \in Y_G^{\square}$ with $s_G(y) = r_1$, defining $H_{\tilde{y}} = \tilde{G}_y$ yields $H{\downarrow}_{\tilde{y}} \cong G{\downarrow}_y \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} G_y \cong H_{\tilde{y}}$.

2. If $p \in \mathcal{T}$, let $y_1 \in Y_G$ be the handle of the step, and $r_1 = s_G(y_1)$ its source.

Defining $H_{\tilde{y}_1}$ by $\overline{H}{\downarrow}_{\tilde{r}_1}$ plus the nodes and edges introduced in the step, gives $H{\downarrow}_{\tilde{y}_1} \underset{\mathcal{E}}{\Longrightarrow} H_{\tilde{y}_1}$ by the correspondence theorem 4.4.

For all boxed edges $y \in Y_G^{\square}$ with $s_G(y) = r_1$, defining $H_{\tilde{y}} = H_{\tilde{y}} \cup (\tilde{G}_{y_1} \backslash \tilde{y}_1)$ gives $H{\downarrow}_{\tilde{y}} \cong G{\downarrow}_y \underset{\mathcal{E}}{\overset{+}{\Longrightarrow}} G_y \underset{\mathcal{E}}{\Longrightarrow} H_{\tilde{y}}$ by the correspondence theorem 4.4.

3. If $p \in \mathcal{L}$, let y_1, y_2, r_1, r_2 be given as in case (1). Defining $H_{\tilde{y}_2} = \tilde{G}_{\tilde{y}_2}$ gives $H{\downarrow}_{\tilde{y}_2} \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} H_{\tilde{y}_1}$ by correctness of G.

For all boxed edges $y \in Y_G^{\square}$ with $s_G(y) = r_1$, defining $H_{\tilde{y}} = H_{\tilde{y}_1} \cup (\tilde{G}_y \backslash \tilde{y})$ yields $H{\downarrow}_{\tilde{y}} \cong G{\downarrow}_y \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} \tilde{G}_y \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} H_{\tilde{y}}$.

Correctness for the remaining edges in $y \in Y_H^{\square}$ is shown by case analysis w.r.t. dependency of $s_G(y)$ from r_1 and r_2. The assumption $\mathcal{E}$ that is non-overlapping makes it easy to show that $G{\downarrow}_y$ is always an $\mathcal{E}$-occurrence. □

4.12 Corollary
If $G \underset{\mathcal{L}}{\Longrightarrow} H$ for correct memo graphs G and H, then $\overline{G} \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} \overline{H}$.

Proof: Consider $G \underset{p}{\Longrightarrow} H$, and let $p \in \mathcal{L}$, $g : L \to G$, and y_1, y_2, r_1, r_2 be given as in case (1) in the proof of theorem 4.11.

Correctness of G implies $G{\downarrow}_{y_2} \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} G_{y_2}$ with $\overline{G}{\downarrow}_{r_2} \subset G_{y_2} \subset \overline{G}$. By definition of $\mathcal{L}$, $\overline{G}{\downarrow}_{r_1} \cong G{\downarrow}_{y_2}$ so that $\overline{G}{\downarrow}_{r_1} \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} G_{y_1}$.

By the join theorem in [Kre77], then $G \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} H'$ with track tr' such that $G_{y_2} \subset H'$ and $\overline{H'}{\downarrow}_{tr'(r_1)} \cong tr'(\overline{G}{\downarrow}_{r_2})$.

This sequence can be extended (by folding the copies of G_{y_2} in H') to a derivation $G \underset{\mathcal{EF}}{\overset{+}{\Longrightarrow}} H' \underset{\mathcal{F}}{\overset{+}{\Longrightarrow}} H''$ with track tr'' such that $tr''(r_1) = tr''(r_2)$ (and thus $H''{\downarrow}_{tr''(r_1)} \cong H''{\downarrow}_{tr''(r_2)}$).

By correctness of lookup steps, then $H'' \cong \overline{H}$. □

We can now state the main result of this section.

4.13 Theorem (Correctness and Completeness)
Let $\mathcal{E}$ be a set of non-looping and non-overlapping evaluation rules.
Then all jungles G_0 satisfy $G_0 \overset{*}{\underset{\mathcal{TFL}}{\Longrightarrow}} H$ iff $G_0 \overset{*}{\underset{\mathcal{EF}}{\Longrightarrow}} H'$ so that $\overline{H} \cong H'$.

Proof: Combine theorems 4.4, 4.11, and 4.12. $\square$

5 Full and Lazy Memoïzation

With the results of the previous section we are able to achieve memoïzation. Let us first recall R.J.M. Hughes' characterization of a memo function in [Hug85]:

> A *memo function* remembers all arguments it is applied to, together with the results computed from them. If it is re-applied to arguments, the function does not re-compute the result, but re-uses the result computed earlier.

More specifically, *full memoïzation* means that results are re-used if the arguments of a function are *equal* to arguments used before; *lazy memoïzation* is restricted to the case that these arguments are *identical*, i.e. represented by the same nodes.

In our approach this is achieved as follows:

- Tabulating steps introduce a boxed edge recording the function applied (its label), its arguments (targets), and its result (source).
- Lookup steps re-use the source (result) of a boxed edge if the function is re-applied to identical arguments.
- Folding steps provide a means to identify nodes representing equal arguments.

In our setting, derivations are memoïzing if a tabulating rule is applied only if no copy of its occurrence has been evaluated before.

5.1 Definition (Full and Lazy Memoïzation)
A derivation step $G \underset{p}{\Longrightarrow} H$ with $p \in \mathcal{TFL}$ and handle $y \in Y_G$ is *lazily memoïzing* (*fully memoïzing*) if $p \in \mathcal{T}$ implies that there is no boxed edge $y' \in Y_G^{\square}$ such that $t_G(y') = t_G(y)$ ($G{\downarrow_{y'}} \overset{*}{\underset{\mathcal{F}}{\Longleftrightarrow}} G{\downarrow_{s_G(y)}}$, resp.). $\square$

In order to meet the requirements above, we define derivation steps where lookup and folding rules have priority.

5.2 Definition (Memoïzation Steps)
We write $G \underset{\mathcal{FL/T}}{\Longrightarrow} H$ (and $G \underset{\mathcal{L/T}}{\Longrightarrow} H$) if either $G \underset{\mathcal{FL}}{\Longrightarrow} H$ ($G \underset{\mathcal{L}}{\Longrightarrow} H$, resp.), or $G \underset{\mathcal{T}}{\Longrightarrow} H$ where G has no $\mathcal{FL}$-occurrence ($\mathcal{L}$-occurrence, resp.). $\square$

5.3 Example
The derivation in figure 4 of example 4.3 consists of $\underset{\mathcal{FL}/\mathcal{T}}{\Longrightarrow}$-steps. □

According to the definitions above, tabulating and lookup rules alone are sufficient to do lazy memoïzation. If folding rules are added, full memoïzation can be achieved.

5.4 Theorem
1. $\underset{\mathcal{L}/\mathcal{T}}{\Longrightarrow}$ is lazily memoïzing.

2. $\underset{\mathcal{FL}/\mathcal{T}}{\Longrightarrow}$ is fully memoïzing.

Proof: Let y be the handle in the derivation.

1. Assuming that some boxed edge $y' \in Y_G^{\square}$ satisfies $t_G(y') = t_G(y)$ leads to the contradiction that y and y' define a $\mathcal{L}$-occurrence in G so that $G \underset{p}{\Longrightarrow} H$ is not lazily memoïzing (and also not fully memoïzing).

2. Assuming that some boxed edge $y' \in Y_G^{\square}$ satisfies $G{\downarrow}_{y'} \overset{*}{\underset{\mathcal{F}}{\Longleftrightarrow}} G{\downarrow}_{s_G(y)}$ allows to conclude that all targets $t_G(y') = v'_1 \dots v'_k$ and $t_G(y) = v_1 \dots v_k$ satisfy $G{\downarrow}_{v'_i} \overset{*}{\underset{\mathcal{F}}{\Longleftrightarrow}} G{\downarrow}_{v_i}$ for $1 \leq i \leq k$. If $t_G(y') = t_G(y)$ we have case *1* above. Otherwise confluence of folding ([HP91, theorem 4.8]) implies for at least one i, $1 \leq i \leq k$ that graph $G{\downarrow}_{v'_i} \overset{*}{\underset{\mathcal{F}}{\Longrightarrow}} G' \overset{*}{\underset{\mathcal{F}}{\Longleftarrow}} G{\downarrow}_{v_i}$ for some jungle G'. Thus $G \underset{p}{\Longrightarrow} H$ is not fully memoïzing. □

Although $\underset{\mathcal{L}/\mathcal{T}}{\Longrightarrow}$ and $\underset{\mathcal{FL}/\mathcal{T}}{\Longrightarrow}$ are strategies w.r.t. the application of of folding and lookup rules, they leave the choice between different $\mathcal{T}$-occurrences within a memo graph completely open. So, these derivations avoid re-evaluation, *independent of a particular evaluation strategy!*

The rules devised can also be used in a more sophisticated and realistic way: Only some of the functions in $\mathcal{O}p$ are considered as *memo functions* for which tabulating and lookup rules are used, wheras evaluation rules are used for the rest. Folding rules can be used, whenever full memoïzation shall be achieved.

In our running example, considering FIB as a (lazy) memo function suffices to improve efficiency.

5.5 Example (Complexity of Fibonacci Function)
Let p denote the evaluation rule of example 3.3, p' the tabulating rule and l the lookup rule in example 4.3.

Consider the $\underset{\mathcal{P}}{\Longrightarrow}$*-complexity* of a jungle G to be the number of steps in the longest $\underset{\mathcal{P}}{\Longrightarrow}$-derivation issuing from G.

Let Fib^n denote the jungle representing the Fibonacci function of n, $n \geq 0$.

Then, the $\underset{p}{\Longrightarrow}$-complexity of Fib^n is well-known to be exponential in n. However, the $\underset{l/p'}{\Longrightarrow}$-complexity of Fib^n is only linear in n! □

In order to estimate the gain of tabulation and lookup, the cost for implementing lookup (and folding) steps has to be taken into account. In principal, these steps require quadratic time to find an occurrence because all pairs of edges have to be checked for equality of operator labels and targets.

However, practical implementations can use hashing techniques to allocate tabulated functions so that duplicates are never created. As the implementation of full memoïzation described in [Kah92] shows, this can bring reasonable practical efficiency.

6 Conclusions

In this paper, we have extended jungle evaluation by memoïzation, an optimization technique for functional programming languages. Jungles are acyclic hypergraphs representing sets of terms with sharing. Jungle evaluation with folding models term rewriting with sharing in a way that important properties of term rewriting systems are preserved (see [HP88] and [HP91]).

We have modified the evaluation rules to *tabulate* the result of a function application, and introduced new *lookup rules* which re-use this result if the function is re-applied to the same arguments. Thus redundant re-evaluation of terms can be avoided.

The model is correct and complete w.r.t. evaluation rules and folding rules and allows to define schemes for *full* and *lazy memoïzation* if the rules are non-overlapping and non-looping.

Further work could try to relax these restrictions as follows:

The *non-overlapping* condition can be relaxed to confluence, if the correctness of a memo graph G is re-phrased as $G{\downarrow_y} \overset{+}{\underset{\mathcal{EF}}{\Longleftrightarrow}} G{\downarrow_{t_G(y)}}$ for all boxed edges y. Then, normalforms are still unique.

The *non-looping* condition can be completely dropped if cyclic representations of *infinte terms* are generally allowed, also on the right hand sides of rules (see, e.g. [DK89]), as do many implementations of graph reduction. See [KKSdV90] for a cyclic graph model of (non-overlapping) rewriting of infinite terms.

Note that folding is of limited use for cyclic graphs as it cannot achieve full collapsing of isomorphic cyclic subgraphs; so this extension works for lazy memoïzation only.

6.1 Example (Infinite Objects)
The paradigm of *computating with infinite objects* is used in lazy functional languages, e.g. for defining infinite lists as in $\mathsf{zeros} = \mathsf{Cons}\ 0\ \mathsf{zeros}$.

Jungle evaluation with (lazy) memoïzation is compatible with this paradigm: See the corresponding tabulating and lookup steps for this example in figure 7 below: Evaluation of this infinite object yields a (cyclic) normal form in only two steps! (Only printing the result will cause looping!) □

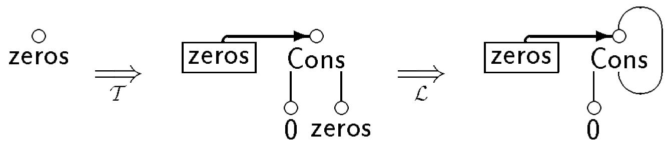


Figure 7: Evaluation of an infinite (cyclic) object (example 6.1)

References

[Der87] N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1,2):69–115, Corrigendum: 4(3), 409–410, 1987.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, chapter 6, pages 243–320. North Holland, 1990.

[DK89] N. Dershowitz and S. Kaplan. Rewrite, Rewrite, Rewrite, Rewrite, Rewrite, ... In *Proc. 16th Symposium on Principles of Programming Languages*, pages 250–259. ACM Press, 1989.

[Ehr79] H. Ehrig. Introduction to the Algebraic Theory of Graph Grammars. In *1st Workshop on Graph Grammars and their Application to Computer Science and Biology*, pages 1–69. Springer, LNCS 73, 1979.

[HKP88] A. Habel, H.-J. Kreowski, and D. Plump. Jungle Evaluation. In *Fifth Workshop on Specification of Abstract Data Types*, pages 92–112. Springer, LNCS 332, 1988. (See also in Fundamenta Informaticae 15(1), 31–60, 1991).

[Hof92] B. Hoffmann. Jungle Evaluation with Memoïzation. Technical Report 2-92, Fachbereich Mathematik / Informatik, Universität Bremen, 1992. *to appear.*

[HP88] B. Hoffmann and D. Plump. Jungle Evaluation for Efficient Term Rewriting. In *Algebraic and Logic Programming*, pages 191–203. Springer, LNCS 343, 1988. *Also in:* Akademie-Verlag Berlin. Long version in technical report 4/88, Fachbereich 3 – Informatik, Universität Bremen.

[HP91] B. Hoffmann and D. Plump. Implementing Term Rewriting by Jungle Evaluation. *RAIRO Theoretical Informatics and Applications*, 25(5):445–472, 1991.

[Hug85] R.J.M. Hughes. Lazy Memo Functions. In Jouannaud, editor, *Conf. on Funct. Lang. and Comp. Arch.*, pages 129–146. IFIP, Springer, LNCS 201, September 1985.

[Kah92] S. Kahrs. Unlimp: Uniqueness as a Leitmotiv for Implementation. In *Proc. PLILP'92 (4th Workshop on Programming Language Implementation and Logic Programming)*. To appear in LNCS, 1992.

[KKSdV90] J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries. Transfinite Reductions in Orthogonal Term Rewriting Systems. Technical Report CS-R9041, Centrum for Wiskunde en Informatica, Amsterdam, 1990. Extended abstract in: *Rewriting Techniques and Applications 1991 (RTA'91)*, R.V. Book, ed., LNCS 488, Springer, 1–12.

[Klo90] J.W. Klop. Term Rewriting Systems. Technical Report CS-R9073, Centre for Mathematics and Computer Science, Amsterdam, 1990. *to appear in:* S. Abramsky, D. Gabbay, and T. Maibaum (eds.): Handbook on Logics in Computer Science, Oxford University Press.

[Klo91] J.W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook on Logics in Computer Science*, volume II, page ??? Oxford University Press, 1991. Early version available as CWI Report CS-R9073, 132 pages, Amsterdam, 1990.

[Kre77] H.-J. Kreowski. *Manipulation von Graphmanipulationen.* PhD thesis, TU Berlin, 1977.

[Pla85] D.A. Plaisted. The Undecidability of self-embedding for term rewriting systems. *Information Processing Letters*, 20:61–64, 1985.