Proceedings Paper:
Plump, DETLEF (1991) Graph-Reducible Term Rewriting Systems. In: Proc. Graph Grammars and Their Application to Computer Science. Lecture Notes in Computer Science. Springer, pp. 622-636.

https://doi.org/10.1007/BFB0017417

Graph-Reducible Term Rewriting Systems *

Detlef Plump
Fachbereich Mathematik und Informatik
Universität Bremen, Postfach 33 04 40
2800 Bremen 33, Germany
det@informatik.uni-bremen.de

Abstract: Term rewriting is commonly implemented by graph reduction in order to improve efficiency. In general, however, graph reduction is not complete: a term may be not normalizable through graph derivations although a normal form exists. Term rewriting systems which permit a complete implementation by graph reduction are called graph-reducible. We show that the following property is sufficient for graph-reducibility: every term having a normal form can be normalized by parallel term rewrite steps in which a rule is applied to *all* occurrences of some subterm. As a consequence, a broad class of term rewriting systems which includes all terminating and all orthogonal systems can be shown to be graph-reducible.

Keywords: term rewriting, graph reduction, completeness of graph reduction

Contents

1. Introduction
2. Jungle Evaluation
3. Graph-Reducibility
4. Reduction Strategies
5. Two Classes of Parallelly Normalizing Systems
6. Conclusion

*Work supported by ESPRIT Basic Research Working Group #3264, *COMPASS*.

1 Introduction

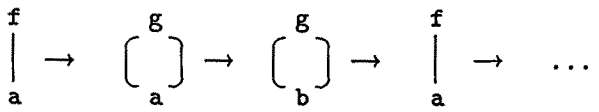
Term rewriting serves as a model of computation in several areas of computing science such as algebraic specification, functional programming, and theorem proving. In practice, implementations of term rewriting usually represent terms by graphs in order to exploit the sharing of common subterms: repeated evaluations of the same subterm are avoided. However, sharing common subterms may lead to incompleteness. As an example, consider the following term rewriting system:

$$\begin{aligned} \mathtt{f(x)} &\to \mathtt{g(x,x)} \\ \mathtt{a} &\to \mathtt{b} \\ \mathtt{g(a,b)} &\to \mathtt{c} \\ \mathtt{g(b,b)} &\to \mathtt{f(a)} \end{aligned}$$

The term $\mathtt{f(a)}$ can be normalized by the rewrite sequence

$$\mathtt{f(a)} \to \mathtt{g(a,a)} \to \mathtt{g(a,b)} \to \mathtt{c}.$$

But no normal form can be reached when common subterms are shared:



This kind of incompleteness could be remedied by introducing graph rules which "unfold" shared subterms. But then the efficiency gained by sharing would get lost, and with it the very advantage of graph reduction. In this paper, we study the class of term rewriting systems that permit completeness despite sharing. The kind of completeness we are aiming at is normalizability: if a graph represents a term that has a normal form, then this graph must be reducible to some normal form. Following Barendregt et al. [BEGKPS 87], we call term rewriting systems with this property *graph-reducible.*

The problem with the term rewriting system shown above is that sharing excludes certain rewrite steps (e.g. $\mathtt{g(a,a)} \to \mathtt{g(a,b)}$). In contrast, we show in section 3 that parallel term rewrite steps of the following kind can always be implemented by graph reduction: a term is rewritten by applying a rule simultaneously to *all* occurrences of some subterm. Such a parallel step can be modelled on the graph level by maintaining a full sharing of common subterms.

Working with parallel term rewrite steps enables us to establish various sufficient conditions for graph-reducibility. In particular, it is straightforward to show that all terminating and all orthogonal term rewriting systems are graph-reducible. Moreover, normalizing term rewriting strategies that are defined in terms of parallel rewrite steps can effectively be translated into normalizing graph reduction strategies. Our classification of graph-reducible systems is complemented by examples which delimit subclasses from each other.

2 Jungle Evaluation

We review the jungle evaluation approach ([HKP 88], [HP 88]) as far as it is necessary for the following sections. We assume that the reader is familiar with basic notions of term rewriting (see for example [DJ 90], [Klo 90]).

Let $SIG = (S, F)$ be a signature, that is, S is a set of sorts and F is a set of function symbols such that each $f \in F$ has a string of argument sorts $s_1 \dots s_n \in S^*$ and a result sort $s \in S$. A function symbol f is written $f : s_1 \dots s_n \to s$ when the argument and result sorts matter.

A *hypergraph* $G = (V_G, E_G, s_G, t_G, l_G, m_G)$ over SIG 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 $l_G : V_G \to S$ and $m_G : E_G \to F$, labeling nodes with sorts and hyperedges with function symbols.

A hypergraph G over SIG is a *jungle* if

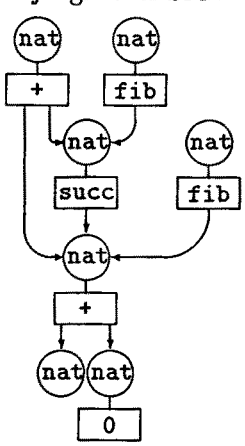
- the labeling of G is compatible with SIG, that is, for each $e \in E_G$, $m_G(e) = f : s_1 \dots s_n \to s$ implies $l_G^*(s_G(e)) = s$ and $l_G^*(t_G(e)) = s_1 \dots s_n$, [1]
- $outdegree_G(v) \leq 1$ for each $v \in V_G$,
- G is acyclic.

2.1 Example

Assume that SIG contains a sort `nat` and function symbols

$$\begin{aligned} &\mathtt{0} :\to \mathtt{nat},\\ &\mathtt{succ}, \mathtt{fib} : \mathtt{nat} \to \mathtt{nat},\\ &+ : \mathtt{nat}\ \mathtt{nat} \to \mathtt{nat}. \end{aligned}$$

Then the following hypergraph is a jungle over SIG.



[1] Given a mapping $g : A \to B$, the extension $g^* : A^* \to B^*$ is defined by $g^*(\lambda) = \lambda$ and $g^*(aw) = g(a)g^*(w)$ for all $a \in A$, $w \in A^*$. Here λ denotes the empty string.

Nodes are drawn as circles and hyperedges as boxes, both with their labels written inside. A line without arrow-head connects a hyperedge 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. □

When *SIG*-terms shall be represented by jungles, nodes without outgoing hyperedge serve as variables. The set of all such nodes in a jungle G is denoted by VAR_G. (Note that each variable in VAR_G occurs only once in G, since variables are nodes rather than labels.) The term represented by a node v is then defined by

$$term_G(v) = \begin{cases} v & \text{if } v \in VAR_G, \\ m_G(e)term^*_G(t_G(e)) & \text{otherwise, where } e \text{ is the unique} \\ & \text{hyperedge with } s_G(e) = v. \end{cases}$$

For instance, if v is the left one of the two topmost nodes in the above example, then $term_G(v) = (\mathtt{x} + 0) + \mathrm{succ}(\mathtt{x} + 0)$ (using infix notation) where $\mathtt{x}$ is the unique variable node in G.

For defining reduction rules and reduction steps we need structure preserving mappings between jungles. A *jungle morphism* $g : G \to H$ consists of two mappings $g_V : V_G \to V_H$ and $g_E : E_G \to E_H$ which preserve sources, targets, and labels, that is, $s_H \circ g_E = g_V \circ s_G$, $t_H \circ g_E = g^*_V \circ t_G$, $l_H \circ g_V = l_G$, and $m_H \circ g_E = m_G$.

Now we are ready to translate term rewrite rules into *evaluation rules*. For motivations and further details, however, we refer to [HP 88].

Assumption
For the rest of this paper we assume that $\mathcal{R}$ is an arbitrary term rewriting system, that is, each rule $l \to r$ in $\mathcal{R}$ 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. □

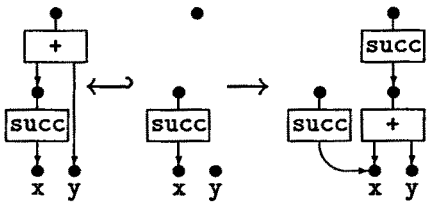
Given a rule $l \to r$ from $\mathcal{R}$, the corresponding evaluation rule $(L \hookleftarrow K \xrightarrow{b} R)$ consists of three jungles L,K,R and jungle morphisms $K \hookrightarrow L$ and $b : K \to R$ such that:

- L is a variable-collapsed tree[2] with $term_L(root_L) = l$.
- K is the subjungle of L obtained by removing the edge outgoing from $root_L$.
- R is constructed from K as follows: if r is a variable, then $root_L$ is identified with the node r; otherwise, R is the disjoint union of K and a variable-collapsed tree R' with $term_{R'}(root_{R'}) = r$ where $root_{R'}$ is identified with $root_L$ and each $x \in VAR_{R'}$ is identified with its counterpart in VAR_K.
- $K \hookrightarrow L$ and $b : K \to R$ are inclusions with the possible exception that b identifies $root_L$ with some variable.

2.2 Example

The following picture shows the evaluation rule for the rewrite rule $\mathrm{succ}(\mathtt{x}) + \mathtt{y} \to \mathrm{succ}(\mathtt{x} + \mathtt{y})$ (where • depicts a nat-labeled node):

[2]That is, there is a unique node $root_L$ with $indegree_L(root_L) = 0$ and only variables may have an indegree greater than 1.



□

We assume from now on that every jungle G has a distinguished node $root_G$ which represents the term to be evaluated. Then the "relevant" part of G consists of all nodes and edges that are reachable from $root_G$. This subjungle is denoted by $G^\bullet$. The nodes and edges in $G - G^\bullet$ are said to be *garbage*. In the following we write $term(G)$ for $term_G(root_G)$.

The application of an evaluation rule $(L \hookleftarrow K \rightarrow R)$ to a jungle G requires to find an occurrence of L in $G^\bullet$ and works then analogously to the construction of K and R described above:

- Find a jungle morphism $g : L \rightarrow G^\bullet$.
- Remove the edge outgoing from $g(root_L)$, yielding a jungle D (with $root_D = root_G$).
- The resulting jungle H is obtained from D like R is constructed from K: if r is a variable, then $g(root_L)$ is identified with $g(r)$; otherwise, H is the disjoint union of D and the variable-collapsed tree R' where $root_{R'}$ is identified with $g(root_L)$ and each $x \in VAR_{R'}$ is identified with $g(x)$. Finally, $root_D$ becomes $root_H$.

We write $G \Rightarrow H$ if H is the result of applying an evaluation rule to G and call this an *evaluation step*. We also use the notation $G \Rightarrow_v H$ if v is the image of $root_L$ in G, that is, if $v = g(root_L)$. For the resulting jungle H its structure and labeling matters rather than the names of (non-variable) nodes. So every jungle H' essentially isomorphic[3] to H is allowed as a result of this evaluation step as well. We require, however, that variable nodes are not renamed.

For stating the effect of an evaluation step $G \Rightarrow H$ on $term(G)$ we need some more notions. Recall that the *positions* in a term t are strings of natural numbers denoting nodes in the tree representation of t and that for each position π in t, $t[\pi]$ is the subterm at π. π is called a *redex* if there is some rule $l \rightarrow r$ and some substitution σ such that $t[\pi] = \sigma(l)$.

Given a nonempty set $\Delta = \{\pi_1, \ldots, \pi_n\}$ of redexes in a term t such that $t[\pi_i] = t[\pi_j]$ for $1 \leq i, j \leq n$, we denote by $t \Vdash_\Delta u$ the parallel rewrite step that reduces all the π_i simultaneously by the same rule.

For every jungle G there is a mapping $node_G$ which takes positions in $term(G)$ to nodes in $G^\bullet$ such that λ is mapped to $root_G$ and each position πi to the i-th target node

[3]Two jungles G and H are said to be *essentially isomorphic* if there is a jungle morphism $g : G^\bullet \rightarrow H^\bullet$ with g_V and g_E bijective. Note that we only compare the non-garbage parts of G and H.

of the unique edge outgoing from $node_G(\pi)$. For each position π, $node_G(\pi)$ represents $term(G)[\pi]$.

2.3 Evaluation Theorem ([HP 88])
Let $G \Rightarrow_v H$ be an evaluation step. Then

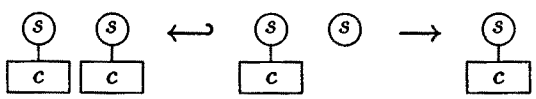
$$term(G) \underset{\Delta}{\Vdash} term(H)$$

with $\Delta = \{\pi \mid node_G(\pi) = v\}$. □

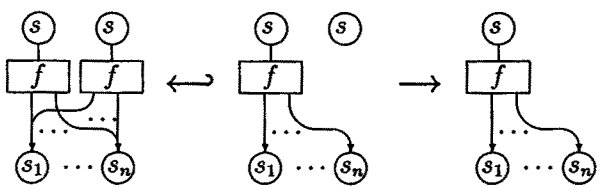
The Evaluation Theorem expresses correctness: evaluation steps perform term rewriting. But there is a problem with evaluation rules for non-left-linear rewrite rules: since the left-hand side L of such a rule is not a tree, there may be no jungle morphism from L to a jungle G although the underlying term rewrite rule is applicable to $term(G)$. This happens if a shared variable in L corresponds to different occurrences of a subterm in G. To overcome this problem we introduce *folding rules* which allow to compress a jungle such that equal subterms are represented by the same node.

For each function symbol in F there is a folding rule $(L \hookleftarrow K \xrightarrow{b} R)$:

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



For each function symbol $f : s_1 \dots s_n \rightarrow s$:



The application of a folding rule to a jungle G is called a *folding step* and works as follows:

- Find a jungle morphism $g : L \rightarrow G^\bullet$ with g_E injective.
- Remove $g(e)$, where e is the unique edge in $L - K$.
- Identify $g(v)$ and $g(v')$, where v and v' are the two source nodes in L.

If H is the resulting jungle, then this folding step is denoted by $G \rightsquigarrow H$. Again we allow every jungle essentially isomorphic to H as a result of this folding step (but require that variables are not renamed). A sequence $G \rightsquigarrow^* \widetilde{G}$ of folding steps is called a *folding*.

A jungle G is said to be *fully collapsed* if no folding rule is applicable to G. Equivalently, G is fully collapsed if for all nodes v_1, v_2 in $G^\bullet$, $term_G(v_1) = term_G(v_2)$ implies $v_1 = v_2$. By applying folding rules as long as possible, every jungle can be transformed into a fully collapsed jungle.

2.4 Theorem ([HP 88])
For every jungle G there is a folding $G \rightsquigarrow^* \tilde{G}$ such that $\tilde{G}$ is fully collapsed. □

Note that folding rules are constructed in such a way that folding preserves terms: $G \rightsquigarrow^* \tilde{G}$ implies $term(G) = term(\tilde{G})$.

In the following we use evaluation and folding rules together and abbreviate $\Rightarrow \cup \rightsquigarrow$ by $\Rrightarrow$. A step $G \Rrightarrow H$ is called a *reduction step* and a sequence $G \Rrightarrow^* H$ of reduction steps is said to be a *reduction.*

A jungle H is a *normal form* (with respect to $\Rrightarrow$) if there is no reduction step $H \Rrightarrow J$. In this case H is a normal form *of* every jungle G with $G \Rrightarrow^* H$. A jungle G *has* a normal form if $G \Rrightarrow^* H$ for some normal form H. These notions are analogously defined for terms (by replacing $\Rrightarrow$ by $\rightarrow$).

2.5 Normal Form Theorem ([HP 88])
Let $G \Rrightarrow^* H$ be a reduction such that H is a normal form. Then $term(H)$ is a normal form of $term(G)$. □

3 Graph-Reducibility

In this section we establish a sufficient condition for term rewriting systems to have a complete implementation by graph reduction. We consider parallel term rewrite steps in which a rule is applied to all occurrences of some subterm. Every sequence of such rewrite steps can be modelled by a sequence of jungle reduction steps in which jungles are completely folded ahead of evaluation steps. As a consequence, a jungle has a normal form if it represents a term that can be normalized by parallel rewriting.

3.1 Definition
A term rewriting system is called *graph-reducible* if for every jungle G, if $term(G)$ has a normal form, then G has a normal form. [4] □

3.2 Theorem
The following is equivalent:

(i) $\mathcal{R}$ is graph-reducible.

(ii) For every jungle G, if $term(G)$ has a normal form, then there is a reduction $G \Rrightarrow^* H$ such that $term(H)$ is a normal form of $term(G)$.

Proof. Let G be a jungle such that $term(G)$ has a normal form.
$(i) \Rightarrow (ii)$. If $\mathcal{R}$ is graph-reducible, then there is a reduction $G \Rrightarrow^* H$ such that H is a normal form. By Theorem 2.5, $term(H)$ is a normal form of $term(G)$.
$(ii) \Rightarrow (i)$. By (ii) there is a reduction $G \Rrightarrow^* H$ such that $term(H)$ is a normal form of $term(G)$. A subsequent folding $H \rightsquigarrow^* \widetilde{H}$ yields a fully collapsed jungle $\widetilde{H}$ which is a

[4] The notion of graph-reducibility was introduced by Barendregt et al. [BEGKPS 87]. Their definition requires that *all* normal forms of a term can be computed by graph reduction. However, the results in [BEGKPS 87] that systems with "hypernormalizing" and "sib-normalizing" strategies are graph-reducible hold only for the weaker definition used here. See also section 4.

normal form by the Evaluation Theorem 2.3. (For if there were any jungle J with $\widetilde{H} \Rightarrow J$, then $term(H) = term(\widetilde{H}) \twoheadrightarrow term(J)$, so $term(H)$ would not be a normal form.) Thus $\mathcal{R}$ is graph-reducible. □

In order to give sufficient conditions for graph-reducibility, we consider the parallel application of a term rewrite rule to all occurrences of some subterm.

3.3 Definition
Given a term t and some redex π in t, $t \rightarrow\!\!\!\!\rightarrow_\pi u$ denotes a parallel rewrite step in which all redexes that represent the same subterm as π are reduced by the same rule. That is, $t \rightarrow\!\!\!\!\rightarrow_\pi u$ if and only if $t \twoheadrightarrow_\Delta u$ with $\Delta = \{\pi' \mid t[\pi'] = t[\pi]\}$. □

By using folding rules it is possible to model arbitrary $\rightarrow\!\!\!\!\rightarrow$-steps by jungle reduction steps. This is achieved by completely folding a jungle before applying an evaluation rule.

3.4 Theorem
For every parallel term rewrite step $t \rightarrow\!\!\!\!\rightarrow u$ and every jungle G with $term(G) = t$ there is a reduction

$$G \overset{*}{\rightsquigarrow} \tilde{G} \Rightarrow H$$

such that $term(H) = u$.

Proof. Let $l \rightarrow r$ be the rewrite rule and σ the substitution associated with $t \rightarrow\!\!\!\!\rightarrow u$. By Theorem 2.4 there is a folding $G \rightsquigarrow^* \tilde{G}$ such that $\tilde{G}$ is fully collapsed. Since $term(\tilde{G}) = t$, there is a unique node v in $\tilde{G}^\bullet$ with $term_{\tilde{G}}(v) = \sigma(l)$. Let $(L \hookleftarrow K \rightarrow R)$ be the evaluation rule for $l \rightarrow r$. As shown in [HP 88], there is a jungle morphism $g : L \rightarrow \tilde{G}^\bullet$ with $g(root_L) = v$, because $\tilde{G}$ is fully collapsed. So there is an evaluation step $\tilde{G} \Rightarrow_v H$. By the Evaluation Theorem 2.3 we have $t = term(\tilde{G}) \twoheadrightarrow_\Delta term(H)$ with $\Delta = \{\pi \mid node_{\tilde{G}}(\pi) = v\}$. $node_{\tilde{G}}$ takes each π with $t[\pi] = \sigma(l)$ to a node representing $\sigma(l)$. Since v is the only node in $\tilde{G}^\bullet$ representing $\sigma(l)$, we conclude that $\Delta = \{\pi \mid t[\pi] = \sigma(l)\}$. Thus $t \rightarrow\!\!\!\!\rightarrow term(H)$ via $l \rightarrow r$ and σ, which implies $term(H) = u$. □

3.5 Corollary
Let $t \rightarrow\!\!\!\!\rightarrow^* u$ for some terms t, u. Then for every jungle G with $term(G) = t$ there is a reduction $G \Rightarrow^* H$ such that $term(H) = u$. □

The above corollary suggests to consider the following kind of term rewriting systems.

3.6 Definition
A term rewriting system is called *parallelly normalizing* if for every term t having a normal form, there is a parallel rewrite sequence $t \rightarrow\!\!\!\!\rightarrow^* u$ to some normal form u. [5] □

Corollary 3.5 together with Theorem 3.2 yields the following result.

[5] Note that "parallelly normalizing" neither implies nor follows from the property that every term has a normal form. A term rewriting system with the latter property is called "normalizing" [DJ 90] or "weakly normalizing" [Klo 90].

3.7 Corollary
Parallelly normalizing term rewriting systems are graph-reducible.

Proof. Let $\mathcal{R}$ be parallelly normalizing and G be a jungle such that $term(G)$ has a normal form. Then $term(G) \rightarrow\!\!\!\!\rightarrow^* u$ for some normal form u. Hence, by Corollary 3.5, there is a reduction $G \Rightarrow^* H$ with $term(H) = u$. With Theorem 3.2 it follows that $\mathcal{R}$ is graph-reducible. □

3.8 Example
Let $\mathcal{R}$ consist of the following three rules for the addition of natural numbers:

$$\begin{aligned} 0 + x &\rightarrow x \\ succ(x) + y &\rightarrow succ(x + y) \\ x + y &\rightarrow y + x \end{aligned}$$

Here a term t has a normal form if and only if there is at most one variable occurrence in t. The following procedure computes this normal form via innermost $\rightarrow\!\!\!\!\rightarrow$-steps (a redex π in t is said to be *innermost* if no proper subterm of $t[\pi]$ contains a redex):

while *there is some innermost redex* π *in* t **do**
begin
  let $t[\pi] = u_1 + u_2$;
  if u_1 *is a variable* **then** $t \rightarrow\!\!\!\!\rightarrow_\pi t'$ *by rule 3*
    else $t \rightarrow\!\!\!\!\rightarrow_\pi t'$ *by rule 1 or 2;*
  $t := t'$
end

The procedure terminates whenever the input term has a normal form, so $\mathcal{R}$ is parallelly normalizing and thus graph-reducible. □

The following example shows that the converse of Corollary 3.7 does not hold, that is, there are graph-reducible term rewriting systems which are not parallelly normalizing.

3.9 Example
Suppose that $\mathcal{R}$ is given as follows:

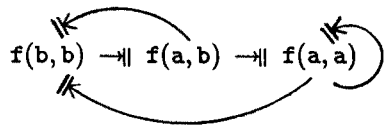
$$\begin{aligned} f(x, y) &\rightarrow f(a, x) \\ a &\rightarrow b \\ f(b, a) &\rightarrow c \end{aligned}$$

To see that $\mathcal{R}$ is graph-reducible, let t_1,t_2 be any terms and G be any jungle with $term(G) = f(t_1, t_2)$. Then

$$G = [f(t_1, t_2)] \Rightarrow [f(a, t_1)] \Rightarrow [f(a, a)] \Rightarrow [f(b, a)] \Rightarrow [c]$$

where, for every term t, $[t]$ denotes some jungle with $term([t]) = t$. Note that the two occurrences of a in $f(a, a)$ are represented by different nodes in $[f(a, a)]$.

Although $\mathcal{R}$ is graph-reducible, it is not parallelly normalizing. $\mathtt{f(b,b)}$, for instance, cannot be normalized by $\twoheadrightarrow\!\!\!\!\!\!\|$-steps:



□

It should be noted that the above example does no longer work if we require that jungles are completely folded before an evaluation rule is applied. Indeed, if we consider graph-reducibility for this special kind of reduction, then a term rewriting system is graph-reducible if and only if it is parallelly normalizing.

4 Reduction Strategies

In evaluating terms or jungles we are interested to perform only steps that eventually lead to a normal form. So we need some control mechanism which selects for every term resp. jungle those steps that have to be performed next. This leads us to the investigation of reduction strategies.

Similar to Barendregt et al. [BEGKPS 87] we use an abstract definition of a strategy which can be applied to both term and graph rewriting.

4.1 Definition
1. Let $(B, \rightarrow)$ be an *abstract reduction system*, that is, B is a set and $\rightarrow$ is a binary relation on B. A *strategy* for $(B, \rightarrow)$ is a function $\mathcal{S}$ that maps each b in B to a set $\mathcal{S}(b)$ satisfying the following two conditions:

- Each element in $\mathcal{S}(b)$ is a reduction sequence of the form $b \rightarrow b_1 \rightarrow b_2 \rightarrow \ldots \rightarrow b_n$ with $n \geq 1$.
- $\mathcal{S}(b)$ is empty only if b is a normal form.

We write $b \rightarrow_{\mathcal{S}} b'$ if there is some sequence $b \rightarrow b_1 \rightarrow \ldots \rightarrow b_n = b'$ in $\mathcal{S}(b)$. $\mathcal{S}$ is called a *one-step strategy* if, for each b in B, all sequences in $\mathcal{S}(b)$ are of length 1. $\mathcal{S}$ is said to be *normalizing* if for each b in B having a normal form, there is no infinite sequence $b \rightarrow_{\mathcal{S}} b_1 \rightarrow_{\mathcal{S}} b_2 \rightarrow_{\mathcal{S}} \ldots$

2. A term rewriting strategy $\mathcal{S}$ is a $\twoheadrightarrow\!\!\!\!\!\!\|$*-strategy* if, for every term t, $\mathcal{S}(t)$ contains only sequences of the form $t \twoheadrightarrow\!\!\!\!\!\!\| t_1 \twoheadrightarrow\!\!\!\!\!\!\| t_2 \twoheadrightarrow\!\!\!\!\!\!\| \ldots \twoheadrightarrow\!\!\!\!\!\!\| t_n$. □

It is straightforward to verify the following characterization of parallelly normalizing systems.

4.2 Fact
A term rewriting system is parallelly normalizing if and only if it has a normalizing $\twoheadrightarrow\!\!\!\!\!\!\|$-strategy. □

By Corollary 3.7 it is in principle possible to compute term normal forms via jungle reductions if $\mathcal{R}$ is parallelly normalizing. To make this reality, however, we have to find a (computable) normalizing jungle reduction strategy. Fortunately, Theorem 3.4 allows us to effectively transform every $\twoheadrightarrow\!\!\!\!\!\!\|$-strategy $\mathcal{S}$ into a jungle strategy $\mathcal{S}_J$.

4.3 Definition
Given some sequence $t \twoheadrightarrow\!\!\!\!\!\!\| t_1 \twoheadrightarrow\!\!\!\!\!\!\| \ldots \twoheadrightarrow\!\!\!\!\!\!\| t_n$ and some jungle G with $term(G) = t$, we denote by $red_G[t \twoheadrightarrow\!\!\!\!\!\!\| t_1 \twoheadrightarrow\!\!\!\!\!\!\| \ldots \twoheadrightarrow\!\!\!\!\!\!\| t_n]$ a jungle reduction $G \Rrightarrow^* G_1 \Rrightarrow^* \ldots \Rrightarrow^* G_n$ with $term(G_i) = t_i$, for $i = 1, \ldots, n$. Such a reduction can be effectively constructed by Theorem 3.4.
Let now G be an arbitrary jungle. $\mathcal{S}_J(G)$ is defined as follows:

- $\mathcal{S}_J(G) = \emptyset$ if G is a normal form.
- $\mathcal{S}_J(G) = \{G \rightsquigarrow^* \tilde{G}\}$ if $term(G)$ is a normal form, but G is not fully collapsed. Here $G \rightsquigarrow^* \tilde{G}$ is some folding such that $\tilde{G}$ is fully collapsed.
- $\mathcal{S}_J(G) = \{red_G[term(G) \twoheadrightarrow\!\!\!\!\!\!\| t_1 \twoheadrightarrow\!\!\!\!\!\!\| \ldots \twoheadrightarrow\!\!\!\!\!\!\| t_n] \mid term(G) \twoheadrightarrow\!\!\!\!\!\!\| t_1 \twoheadrightarrow\!\!\!\!\!\!\| \ldots \twoheadrightarrow\!\!\!\!\!\!\| t_n \in \mathcal{S}(term(G))\}$ if neither of the first two cases applies.

4.4 Theorem
If $\mathcal{S}$ is a normalizing $\twoheadrightarrow\!\!\!\!\!\!\|$-strategy, then $\mathcal{S}_J$ is normalizing.

Proof. Let G be a jungle having a normal form. Suppose there were an infinite sequence $G \Rrightarrow_{\mathcal{S}_J} G_1 \Rrightarrow_{\mathcal{S}_J} G_2 \Rrightarrow_{\mathcal{S}_J} \ldots$ Then $term(G) \to_{\mathcal{S}} term(G_1) \to_{\mathcal{S}} term(G_2) \to_{\mathcal{S}} \ldots$ by the above construction. But $term(G)$ has a normal form by Theorem 2.5, so $\mathcal{S}$ would not be normalizing. □

If some term rewriting strategy $\mathcal{S}$ is known to be normalizing, one may try to "parallelize" $\mathcal{S}$ in order to obtain a normalizing $\twoheadrightarrow\!\!\!\!\!\!\|$-strategy. There is a natural choice for such a transformation if $\mathcal{S}$ is a one-step strategy.

4.5 Definition
Let $t \to u$ be a term rewrite step via some rule $l \to r$ and some redex π. Then $t \twoheadrightarrow\!\!\!\!\!\!\|_\pi \hat{u}$ via $l \to r$ is called the *parallelization* of $t \to u$. For every one-step strategy $\mathcal{S}$, the $\twoheadrightarrow\!\!\!\!\!\!\|$-strategy $\mathcal{S}_\|$ is defined by

$$\mathcal{S}_\|(t) = \{t \twoheadrightarrow\!\!\!\!\!\!\| \hat{u} \mid t \twoheadrightarrow\!\!\!\!\!\!\| \hat{u} \text{ is the parallelization of some step } t \to u \text{ in } \mathcal{S}(t)\}.$$

□

Examples for one-step strategies that are normalizing after parallelization are the so-called sib-normalizing strategies considered by Barendregt et al. [BEGKPS 87]. A one-step strategy $\mathcal{S}$ is said to be *sib-normalizing* if the extended strategy $\tilde{\mathcal{S}}$ defined by

$$\tilde{\mathcal{S}}(t) = \{t \underset{\mathcal{S}}{\to} t' \overset{*}{\to} t'' \mid t' \overset{*}{\to} t'' \text{ reduces siblings of the redex reduced by } t \underset{\mathcal{S}}{\to} t'\}$$

is normalizing. (A *sibling* of a redex π in a term t is a redex π' with $t[\pi] = t[\pi']$.) In [BEGKPS 87] it is shown that every left-linear term rewriting system that has a sib-normalizing strategy is graph-reducible.

4.6 Theorem
Every term rewriting system that has a sib-normalizing strategy is parallelly normalizing. The converse does not hold.

Proof. It is evident that $t \to_{\mathcal{S}_\parallel} t'$ implies $t \to_{\widetilde{\mathcal{S}}} t'$, for every one-step strategy $\mathcal{S}$ and for all terms t, t'. Hence if $\mathcal{S}$ is sib-normalizing, then $\mathcal{S}_\parallel$ is a normalizing $\twoheadrightarrow_\parallel$-strategy. To show that the converse is not true, let $\mathcal{R}$ consist of the two rules $\mathtt{a} \to \mathtt{b}$ and $\mathtt{a} \to \mathtt{f(a,a)}$ [6]. Then the $\twoheadrightarrow_\parallel$-strategy which applies the first rule simultaneously to all occurrences of a in a term is normalizing. But for every one-step strategy $\mathcal{S}$ and for every term t that contains at least two occurrences of a, there is a step $t \to_{\widetilde{\mathcal{S}}} t'$ such that t' again contains two occurrences of a. This is because after a step $t \to_{\mathcal{S}} \overline{t}$ there is an occurrences of a in $\overline{t}$ to which the second rule can be applied. Hence $\widetilde{\mathcal{S}}$ is not normalizing. □

An interesting difference between $\widetilde{\mathcal{S}}$ and $\mathcal{S}_\parallel$ is that (by definition) for $\widetilde{\mathcal{S}}$ to be normalizing it is necessary that $\mathcal{S}$ is normalizing, while $\mathcal{S}_\parallel$ may be normalizing even if $\mathcal{S}$ is not. As an example for the latter, consider two terminating term rewriting systems $\mathcal{R}_0$ and $\mathcal{R}_1$ with disjoint function symbols. Then $\mathcal{R}_0 \cup \mathcal{R}_1$ may be non-terminating. In this case the strategy $\mathcal{S} : t \mapsto \{t \to t' \mid t' \text{ is an arbitrary term}\}$ is not normalizing (and hence not sib-normalizing) while $\mathcal{S}_\parallel$ is normalizing (cf. section 5.1).

5 Two Classes of Parallelly Normalizing Systems

In this section we show that both terminating and orthogonal term rewriting systems are parallelly normalizing. (In fact, the termination property can be relaxed to "parallel termination"). This gives us practical methods to prove graph-reducibility: many methods for proving termination are known (see [Der 87]), and orthogonality is a syntactic property which is simple to check.

5.1 Parallelly Terminating Systems

A term rewriting system is *terminating* if there is no infinite rewrite sequence $t_1 \to t_2 \to t_3 \to \ldots$ Clearly, in this case there is neither an infinite sequence of $\twoheadrightarrow_\parallel$-steps, so every terminating system is parallelly normalizing. This argument suggests the following weaker condition.

5.1 Definition
A term rewriting system is called *parallelly terminating* if there is no infinite sequence $t_1 \twoheadrightarrow_\parallel t_2 \twoheadrightarrow_\parallel t_3 \twoheadrightarrow_\parallel \ldots$ □

Hence for parallelly terminating systems every $\twoheadrightarrow_\parallel$-strategy is normalizing. The following example (from Toyama [Toy 87]) demonstrates that parallelly terminating systems are not necessarily terminating:

[6] This example is due to Michael Löwe.

$$\begin{aligned}\mathcal{R}_0: &\ \mathtt{f(0,1,x)} \to \mathtt{f(x,x,x)}\\ \mathcal{R}_1: &\ \mathtt{or(x,y)} \to \mathtt{x}\\ &\ \mathtt{or(x,y)} \to \mathtt{y}\end{aligned}$$

Although $\mathcal{R}_0$ and $\mathcal{R}_1$ are terminating, $\mathcal{R}_0 \cup \mathcal{R}_1$ is non-terminating since the term `f(or(0,1),or(0,1),or(0,1))` reduces in three steps to itself. In [Plu 91] it is shown that jungle reduction is terminating for $\mathcal{R}_0 \cup \mathcal{R}_1$, that is, there is no infinite sequence of $\Rightarrow$-steps. With Theorem 3.4 it follows that $\mathcal{R}_0 \cup \mathcal{R}_1$ is parallelly terminating. Actually, the main result in [Plu 91] implies that a combined system $\mathcal{R}_0 \cup \mathcal{R}_1$ is parallelly terminating whenever $\mathcal{R}_0$ and $\mathcal{R}_1$ are terminating and when the left-hand sides of $\mathcal{R}_i$ have no common function symbols with the right-hand sides of $\mathcal{R}_{1-i}$ $(i = 0, 1)$.

5.2 Orthogonal Systems

Orthogonal (or *regular*) term rewriting systems are characterized by two properties: their rules are *left-linear* and *nonoverlapping*. That is, every variable occurs at most once in every left-hand side, and whenever there are two rules $l \to r$ and $l' \to r'$ such that l overlaps l' in some subterm u, then $l = u$ and $l \to r = l' \to r'$. (A term t *overlaps* a term t' in a subterm u of t, if u is not a variable and there are substitutions σ and σ' such that $\sigma(u) = \sigma'(t')$.)

An example for an orthogonal term rewriting system is *Combinatory Logic*:

$$\begin{aligned}&((\mathtt{S} \cdot \mathtt{x}) \cdot \mathtt{y}) \cdot \mathtt{z} \to (\mathtt{x} \cdot \mathtt{z}) \cdot (\mathtt{y} \cdot \mathtt{z})\\ &(\mathtt{K} \cdot \mathtt{x}) \cdot \mathtt{y} \to \mathtt{x}\\ &\mathtt{I} \cdot \mathtt{x} \to \mathtt{x}\end{aligned}$$

The (nullary) function symbols `S`, `K`, `I` denote combinators while the binary function symbol $\cdot$ denotes function application. (See [Klo 90] for the importance of Combinatory Logic.)

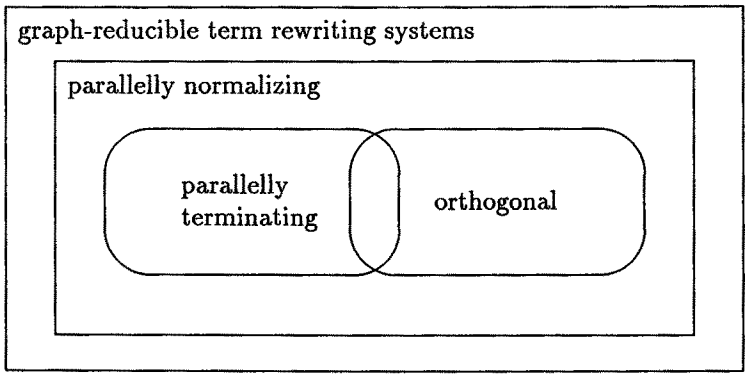
It is well-known that the *parallel outermost* strategy is normalizing for orthogonal systems. This strategy reduces all the outermost redexes in a term in parallel. By a slight modification of the parallel outermost strategy we obtain a normalizing $\to\!\!\!|\!|$-strategy. Let t be any term not in normal form, and let $\Pi = \{\pi_1, \ldots, \pi_n\}$ be a set of outermost redexes in t such that

- $t[\pi_i] \neq t[\pi_j]$ for $i \neq j$, and
- for each outermost redex π in t there is some $\pi_i \in \Pi$ with $t[\pi] = t[\pi_i]$.

Then $\mathcal{S}(t) = \{t \to\!\!\!|\!|_{\pi_1} t_1 \to\!\!\!|\!|_{\pi_2} \ldots \to\!\!\!|\!|_{\pi_n} t_n\}$ defines a $\to\!\!\!|\!|$-strategy. (Note that $\mathcal{S}$ is well-defined by orthogonality: π_{i+1} is a redex in t_i for $i = 1, \ldots, n-1$.) $\mathcal{S}$ is normalizing by O'Donnell's result that "eventually outermost" rewrite sequences are normalizing (see [O'D 77]).

6 Conclusion

We can summarize our classification of graph-reducible term rewriting systems as follows:



All the inclusions in this diagram are proper. (Example 3.8 shows a parallelly normalizing system which is neither parallelly terminating nor orthogonal.) Moreover, the example shown in the introduction demonstrates that confluence[7] is not sufficient for graph-reducibility.

Finally, we mention some topics for further research:

- Not all graph-reducible systems are parallelly normalizing. Is there a satisfactory *characterization* of graph-reducibility in terms of term rewriting notions?
- Are there sufficient *syntactic* conditions for graph-reducibility which go beyond orthogonality? If so, do these conditions give rise to efficient (or at least computable) normalizing strategies?
- Our definition of graph-reducibility can be strengthened by requiring that for every jungle G *all* normal forms of $term(G)$ can be computed by jungle reductions. This *strong graph-reducibility* clearly holds for graph-reducible systems in which every term has at most one normal form, so in particular for orthogonal systems. But what can be said for other classes of term rewriting systems? Terminating systems, for instance, are not strongly graph-reducible in general (a counterexample can be found in the long version of [HP 88]).

Acknowledgements
I wish to thank Berthold Hoffmann and Michael Löwe for comments on a preliminary version of this paper.

[7] A term rewriting system is called *confluent* if for all terms t,t_1,t_2 with $t_1 \,{}^*\!\leftarrow t \rightarrow^* t_2$ there is a term t_3 such that $t_1 \rightarrow^* t_3 \,{}^*\!\leftarrow t_2$.

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 87] N. Dershowitz: *Termination of Rewriting.* Journal of Symbolic Computation, vol. 3, no. 1/2, 69-115 (1987). Corrigendum: vol. 4, no. 3, 409-410

[DJ 90] N. Dershowitz, J.-P. Jouannaud: *Rewrite Systems.* In J. van Leeuwen (ed.): *Handbook of Theoretical Computer Science.* Vol. B, North-Holland (1990)

[HKP 88] A. Habel, H.-J. Kreowski, D. Plump: *Jungle Evaluation.* Proc. Fifth Workshop on Specification of Abstract Data Types, Lecture Notes in Comp. Sci. 332, 92-112 (1988)

[HP 88] B. Hoffmann, D. Plump: *Jungle Evaluation for Efficient Term Rewriting.* Proc. Algebraic and Logic Programming, Akademie-Verlag, Berlin, 191-203 (1988). Also in Lecture Notes in Comp. Sci. 343 (1989). Long version to appear in RAIRO Theoretical Informatics and Applications

[Klo 90] J.W. Klop: *Term Rewriting Systems — From Church-Rosser to Knuth-Bendix and Beyond.* Proc. ICALP '90, Lecture Notes in Comp. Sci. 443, 350-369 (1990)

[O'D 77] M.J. O'Donnell: *Computing in Systems Described by Equations.* Lecture Notes in Comp. Sci. 58 (1977)

[Plu 91] D. Plump: *Implementing Term Rewriting by Graph Reduction: Termination of Combined Systems.* Proc. 2nd Int. Workshop on Conditional and Typed Rewriting Systems, Montreal 1990, to appear in Lecture Notes in Comp. Sci.

[Toy 87] Y. Toyama: *Counterexamples to Termination for the Direct Sum of Term Rewriting Systems.* Information Process. Lett. 25, 141-143 (1987)