The version of the following full text has not yet been defined or was untraceable and may differ from the publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/18699

Please be advised that this information was generated on 2019-08-26 and may be subject to change.
Term Graph Rewriting

Detlef Plump

Computing Science Institute/

CSI-R9822 September 1998
Term Graph Rewriting*

Detlef Plump
Universität Bremen
Fachbereich Mathematik und Informatik
Postfach 33 04 40, 28334 Bremen, Germany
det@informatik.uni-bremen.de

Abstract

Term graph rewriting is concerned with the representation of functional expressions as graphs, and the evaluation of these expressions by rule-based graph transformation. Representing expressions as graphs allows to share common subexpressions, improving the efficiency of term rewriting in space and time. Besides efficiency, term graph rewriting differs from term rewriting in properties like termination and confluence. This paper is a survey of (acyclic) term graph rewriting, where emphasis is given to the relations between term and term graph rewriting. We focus on soundness of term graph rewriting with respect to term rewriting, on completeness for proving validity of equations and for computing term normal forms, on termination and confluence, and on term graph narrowing.

Keywords: term graph rewriting, termination, confluence, term rewriting, narrowing

Classification: 68Q05, 68Q40, 68Q42 (AMS '91); D.1.1, F.1.1, F.4.2, I.1.1 (CR '98)

Note: This paper will appear in Handbook of Graph Grammars and Computing by Graph Transformation, volume 2, editors H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg; World Scientific.

*This paper was partly written while the author was visiting the Computing Science Institute of the University of Nijmegen. Support by Deutsche Forschungsgemeinschaft is gratefully acknowledged.
## Contents

1 Introduction ................................................................. 4

2 Abstract Reduction Systems ............................................. 6

3 Term Graphs ...................................................................... 9

3.1 From hypergraphs to term graphs ................................. 9

3.2 Collapsing, copying and bisimilarity ............................. 12

3.3 Bibliographic notes ....................................................... 14

4 Term Graph Rewriting ..................................................... 16

4.1 Term rewriting .......................................................... 16

4.2 Term graph rewriting .................................................. 17

4.3 Incorporating collapsing and copying ............................ 20

4.4 Bibliographic notes ....................................................... 23

5 Completeness .................................................................. 24

5.1 Simulating arbitrary term rewrite derivations ............... 24

5.2 Graph-reducibility ....................................................... 27

5.3 Bibliographic notes ....................................................... 29

6 Termination .................................................................... 30

6.1 The relation to term rewriting ..................................... 30

6.2 Combined systems ..................................................... 32

6.3 A recursive path order on term graphs ........................ 34

7 Confluence ..................................................................... 38

7.1 The relation to term rewriting ..................................... 38

7.2 Decidability and combined systems ............................. 41
8 Term Graph Narrowing

8.1 Term graph narrowing ......................................................... 46
8.2 Minimally and maximally collapsing narrowing .................. 50
8.3 Bibliographic notes ............................................................. 53

9 Further Topics

References
1 Introduction

Term graph rewriting is concerned with the representation of functional expressions as graphs, and the evaluation of these expressions by rule-based graph transformation. Representing expressions as graphs is motivated by efficiency considerations. Consider, for example, the following rewrite rules for defining multiplication of natural numbers (where $s$ denotes the successor function on natural numbers):

$$
\begin{align*}
  x \times 0 & \rightarrow 0 \\
  x \times s(y) & \rightarrow (x \times y) + x
\end{align*}
$$

In applying the second rule to an expression of the form $t \times s(u)$, the subexpression $t$ has to be copied. This is conspicuous when expressions are drawn as trees:

Copying $t$, however, is expensive in space and time if $t$ is a large expression. Even worse, if $t$ is not yet evaluated, all the work necessary to evaluate it is duplicated by the above rewrite step.

An obvious solution to this problem is, instead of copying $t$, to create two pointers to the existing subexpression $t$. The above rewrite step looks then as follows:

The resulting graph is called a term graph, and the unique occurrence of $t$ is said to be shared. Evaluating this occurrence will correspond to a parallel evaluation
of the two occurrences of \( t \) in the expression \((t \times u) + t\). Thus, sharing subexpressions saves not only space but also time that otherwise would be wasted in repeatedly evaluating equal subexpressions.

Rewriting term graphs rather than expressions, which come as strings or trees, has several consequences besides efficiency issues. This is because certain rewrite sequences are prevented when subexpressions are shared. For example, it may be possible to apply two different rules, at the same position, to the two occurrences of \( t \) in \((t \times u) + t\). These two independent steps are impossible if \( t \) is shared. As a result, term graph rewriting may fail to transform an expression into an irreducible form. (This does not happen with the above rules, though.) Moreover, we will see that term graph rewriting differs from conventional term rewriting in properties like termination and confluence.

This paper intends to be a survey of term graph rewriting, where the scope is restricted to acyclic term graphs. Dealing with acyclic graphs allows to relate term graph rewriting with the rich theory of term rewriting. (See the textbook [11] for a comprehensive survey of term rewriting.) In fact, acyclic term graph rewriting can be seen as a sound implementation of term rewriting, which more accurately reflects the properties of real implementations. Application areas of term and term graph rewriting include theorem proving, functional and logic programming, software specification, and computer algebra.

Our presentation stresses the relations between term and term graph rewriting. We focus on soundness of term graph rewriting with respect to term rewriting, on completeness for proving validity of equations and for computing term normal forms, on termination and confluence, and on term graph narrowing. Some further topics are briefly mentioned in Section 9. To keep this survey concise, proofs are given only occasionally.

Acknowledgements. The author wishes to thank Oliver Dressler, Annegret Habel, Berthold Hoffmann, Jan Willem Klop and Jürgen Müller for providing comments on a previous version of this paper.
2 Abstract Reduction Systems

Rewriting systems (also called reduction or replacement systems) are means to compute by a stepwise transformation of objects. These objects may be strings, terms, formulas, graphs or any other entities from a given domain. The present paper is concerned with rewriting systems over the domains of terms and term graphs. Several concepts and properties of rewriting systems can be defined and studied independently from specific domains, having the advantage that abstract properties of relations can be separated from properties depending on the structure of objects.

Abstract reduction systems are sets together with a binary relation representing elementary transformation steps. They were studied the first time by Newman [80] and systematically applied in influential papers of Rosen [93] and Huet [52]. In the following some basic notions and facts for abstract reduction systems are collected. Further concepts and results can be found, for example, in [9, 11, 16, 28, 52, 56, 66]. As the terminology in the literature is not completely uniform, alternative terms are given in parentheses.

Definition 2.1 (Abstract reduction system)
An abstract reduction system \( (A, \rightarrow) \) consists of a set \( A \) and a binary relation \( \rightarrow \) on \( A \).

For the rest of this section, let \( (A, \rightarrow) \) be an arbitrary abstract reduction system. Given two elements \( a \) and \( b \) in \( A \) such that \( (a, b) \in \rightarrow \), this is denoted by \( a \rightarrow b \).

The inverse relation of \( \rightarrow \) is denoted by \( \leftarrow \), and the composition of two binary relations \( \rightarrow_1 \) and \( \rightarrow_2 \) on \( A \) is written \( \rightarrow_1 \circ \rightarrow_2 \).

Definition 2.2
(1) The identity on \( A \) is the relation \( \rightarrow^0 = \{ (a, a) \mid a \in A \} \).

(2) The reflexive closure of \( \rightarrow \) is the relation \( \rightarrow^\ast = \rightarrow \cup \rightarrow^0 \).

(3) For every \( n > 0 \), the \( n \)-fold composition of \( \rightarrow \) is the relation \( \rightarrow^n = \circ \rightarrow^{n-1} \).

(4) The transitive closure of \( \rightarrow \) is the relation \( \rightarrow^+ = \bigcup_{n>0} \rightarrow^n \).

(5) The transitive-reflexive closure of \( \rightarrow \) is the relation \( \rightarrow^\ast = \rightarrow^+ \cup \rightarrow^0 \).

(6) The symmetric closure of \( \rightarrow \) is the relation \( \leftrightarrow = \rightarrow \cup \leftarrow \).

(7) The equivalence closure of \( \rightarrow \) (or convertibility with respect to \( \rightarrow \)), denoted by \( \leftrightarrow^\ast \), is the transitive-reflexive closure of \( \leftrightarrow \).
Two elements $a$ and $b$ are *convertible* if $a \leftrightarrow^* b$, and they have a *common reduct* if $a \rightarrow^* c \leftarrow^* b$ for some $c$. An element $a$ is a *normal form* if there is no $b$ such that $a \rightarrow b$, and it has a normal form if $a \rightarrow^* b$ for some normal form $b$. In the latter case $b$ is a normal form of $a$.

**Definition 2.3 (Termination and confluence properties)**

The relation $\rightarrow$ is

1. **terminating** (or *strongly normalizing* or *noetherian*) if there does not exist an infinite sequence of the form $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \ldots$,

2. **normalizing** (or *weakly normalizing*) if each element in $A$ has a normal form,

3. **Church-Rosser** if for all $a$ and $b$ with $a \leftrightarrow^* b$ there is some $c$ such that $a \rightarrow^* c \leftarrow^* b$ (see Figure 1(a)),

4. **confluent** if for all $a$, $b$ and $c$ with $b \leftarrow^* a \rightarrow^* c$ there is some $d$ such that $b \rightarrow^* d \leftarrow^* c$ (see Figure 1(b)),

5. **locally confluent** (or *weakly confluent*) if for all $a$, $b$ and $c$ with $b \leftarrow a \rightarrow c$ there is some $d$ such that $b \rightarrow^* d \leftarrow^* c$ (see Figure 1(c)),

6. **subcommutative** if for all $a$, $b$ and $c$ with $b \leftarrow a \rightarrow c$ there is some $d$ such that $b \rightarrow^* d \leftarrow^* c$ (see Figure 1(d)),

7. **convergent** if it is terminating and confluent.

**Lemma 2.4**

1. Termination implies normalization.

2. The Church-Rosser property is equivalent to confluence.

3. Subcommutativity implies confluence.


5. Confluence implies uniqueness of normal forms, that is, whenever $a \leftrightarrow^* b$ for normal forms $a$ and $b$, then $a = b$.

**Proof**

The implications (1) and (4) are obvious. As to (2), the Church-Rosser property clearly implies confluence, and the converse is shown by induction on the number of $\leftrightarrow$-steps constituting an equivalence $a \leftrightarrow^* b$. Statement (3) is proved by two inductions, the first showing that if $\rightarrow$ is subcommutative, then for all $a$, $b$ and $c$ with $b \leftarrow a \rightarrow^* c$ there is some $d$ such that $b \rightarrow^* d \leftarrow^* c$, while the second induction shows that the latter property implies confluence. Finally, it is easy to see that (5) follows from (2).
Notice that by statement (5), in a confluent relation every element has at most one normal form. The converses of the implications (1), (3), (4) and (5) do not hold. Figure 2 shows a well-known counterexample to the converse of (4), viz. a locally confluent (and normalizing) relation that is not confluent. By the following result, however, local confluence and confluence are equivalent in the presence of termination. (See [52] for a short proof of this fact.)

**Lemma 2.5 (Newman’s Lemma [80])**

A terminating relation is confluent if and only if it is locally confluent.
3 Term Graphs

Graphs that represent expressions can be defined in various ways. Here we use acyclic hypergraphs where hyperedges are labelled with function symbols and variables. Each node in such a term graph represents a well-formed expression, a term. We will see that for every term, the set of all its term graph representations forms a complete lattice under a suitable partial order.

3.1 From hypergraphs to term graphs

Let $\Sigma$ be a set of function symbols where each $f \in \Sigma$ comes with a natural number $\text{arity}(f) \geq 0$. Function symbols of arity 0 are called constants. Let further $X$ be an infinite set of variables such that $X \cap \Sigma = \emptyset$, and define $\text{arity}(x) = 0$ for each $x \in X$.

A hypergraph over $\Sigma$ and $X$ is a system $G = (V_G, E_G, \text{lab}_G, \text{att}_G)$ consisting of two finite sets $V_G$ and $E_G$ of nodes (or vertices) and hyperedges, a labelling function $\text{lab}_G : E_G \rightarrow \Sigma \cup X$, and an attachment function $\text{att}_G : E_G \rightarrow V_G^*$ assigning a string of nodes to a hyperedge $e$ such that the length of $\text{att}_G(e)$ is $1 + \text{arity}(\text{lab}_G(e))$. In the sequel, hypergraphs and hyperedges are simply called graphs and edges.

Given an edge $e$ with $\text{att}_G(e) = v_0 \ldots v_n$, node $v_0$ is the result node of $e$ while $v_1, \ldots, v_n$ are the argument nodes. The result node and the (possibly empty) string $v_1 \ldots v_n$ are denoted by $\text{res}(e)$ and $\text{arg}(e)$, respectively.

A path is an alternating sequence $(v_0, i_1, v_1, i_2, \ldots, i_n, v_n)$ ($n \geq 0$) of nodes and positive integers such that for $j = 1, \ldots, n$, if $e$ is the unique edge with $\text{res}(e) = v_{j-1}$, then $v_j$ is the $i_j$-th node in $\text{arg}(e)$. We say that this is a path from $v_0$ to $v_n$.

A node $v'$ is reachable from a node $v$ if there is a path from $v$ to $v'$. A graph is acyclic if it does not contain a path in which some node occurs twice.

Definition 3.1 (Term graph)
A graph $G$ is a term graph if

(1) there is a node $\text{root}_G$ from which each node is reachable,

(2) $G$ is acyclic, and

(3) each node is the result node of a unique edge.

Figure 3 shows a term graph with binary function symbols $+$ and $\times$, a unary function symbol $s$, a constant 0 and a variable $y$. In the representation on the
left, edges are depicted as boxes with inscribed labels, and bullets represent nodes. A line connects each edge with its result node, while arrows point to the argument nodes. The left-to-right order of the arrows leaving the box corresponds to the order of the argument nodes.

On the right of Figure 3, the same term graph is depicted in an alternative, more compact way. In the following both formats will be used to represent term graphs graphically.

![Figure 3: A term graph, depicted in two different ways](image)

A term over $\Sigma$ and $X$ is a variable, a constant, or a string $f(t_1,\ldots,t_n)$ where $f$ is a function symbol of arity $n \geq 1$ and $t_1,\ldots,t_n$ are terms. The subterms of a term $t$ are $t$ and, if $t = f(t_1,\ldots,t_n)$, all subterms of $t_1,\ldots,t_n$.

**Definition 3.2 (Term representation)**

A node $v$ in a term graph $G$ represents the term

$$\text{term}_G(v) = \text{lab}_G(e)(\text{term}_G(v_1),\ldots,\text{term}_G(v_n)),$$

where $e$ is the unique edge with $\text{res}(e) = v$, and where $\text{arg}(e) = v_1\ldots v_n$. It is understood that if $\text{arg}(e)$ is empty, this means $\text{term}_G(v) = \text{lab}_G(e)$. We denote $\text{term}_G(\text{root}_G)$ also by $\text{term}(G)$.

Note that the recursion in the above definition ends because term graphs are acyclic. For example, if $G$ is the term graph of Figure 3, then

$$\text{term}(G) = + (s(0), +( \times (s(0), + (0, y)), \times (s(0), + (0, y)))).$$
Using infix notation for + and \(\times\), this term reads

\[ s(0) + ((s(0) \times (0 + y)) + (s(0) \times (0 + y))). \]

A graph morphism \(f: G \rightarrow H\) between two graphs \(G\) and \(H\) consists of two functions \(f_V: V_G \rightarrow V_H\) and \(f_E: E_G \rightarrow E_H\) that preserve labels and attachment to nodes, that is, \(\text{lab}_H \circ f_E = \text{lab}_G\) and \(\text{att}_H \circ f_E = f_V \circ \text{att}_G\) (where \(\circ\) denotes function composition and \(f^*_V : V_G \rightarrow V_H^*\) maps a string \(v_0 \ldots v_n\) to \(f_V(v_0) \ldots f_V(v_n)\)). The morphism \(f\) is injective (surjective) if \(f_V\) and \(f_E\) are. If \(f\) is injective and surjective, then it is an isomorphism. In this case \(G\) and \(H\) are isomorphic, which is denoted by \(G \cong H\).

Usually one does not want to distinguish between isomorphic (term) graphs. For example, it is more convenient to deal with the tree representation of a term than with an infinite class of isomorphic trees, and it is easier to handle confluence than “confluence up to isomorphism”. To achieve this technically, one may work with isomorphism classes of term graphs, but then one loses access to nodes and edges. We pursue an alternative solution and introduce standard term graphs which serve as unique representatives of their isomorphism classes. The idea is to number the nodes of a term graph in a canonical way, similar to the numbering of positions in terms (see for example [11]).

Given a node \(v\) in a term graph \(G\), an access path of \(v\) [4] is a possibly empty sequence of positive integers \((i_1, i_2, \ldots, i_n)\) such that there exists a path \((v_0, i_1, v_1, i_2, \ldots, i_n, v_n)\) with \(v_0 = \text{root}_G\) and \(v_n = v\). We denote by \(\text{Acc}(v)\) the set of all access paths of \(v\).

**Definition 3.3 (Standard term graph)**

A term graph \(G\) is a standard term graph if

1. \(v = \text{Acc}(v)\) for each node \(v\), and
2. \(e = \text{res}(e)\) for each edge \(e\).

An example for the naming of nodes in a standard term graph is given in Figure 4.

For every term graph we can construct an isomorphic standard term graph by replacing each node \(v\) with \(\text{Acc}(v)\) and modifying the edge set and the labelling and attachment functions correspondingly. Moreover, given a graph morphism \(f: G \rightarrow H\) between term graphs such that \(f_V(\text{root}_G) = \text{root}_H\), we have \(\text{Acc}(v) \subseteq \text{Acc}(f_V(v))\) for each node \(v\) in \(G\). This implies the following property, showing that every isomorphism class of term graphs contains exactly one standard term graph.
Lemma 3.4  
For all standard term graphs $G$ and $H$, $G \cong H$ if and only if $G = H$.

From now on we will tacitly assume that we are dealing with standard term graphs only.

3.2 Collapsing, copying and bisimilarity

By the three conditions of Definition 3.1, a graph morphism $f : G \to H$ between term graphs $G$ and $H$ is surjective if and only if $f_V(\text{root}_G) = \text{root}_H$. In this case $H$ can be seen as a “collapsed” or “compressed” version of $G$.

Definition 3.5 (Collapsing and copying)  
Given two term graphs $G$ and $H$, $G$ collapses to $H$ if there is a graph morphism $G \to H$ mapping $\text{root}_G$ to $\text{root}_H$. This is denoted by $G \succeq H$ or, if the morphism is non-injective, by $G \succ H$. The latter kind of collapsing is said to be proper. The inverse relation of collapsing is called copying and is denoted by $\preceq$. Proper copying, denoted by $\prec$, is the inverse relation of proper collapsing.

Two examples of collapsing and copying are given in Figure 5.

Lemma 3.6  
For all term graphs $G$ and $H$, $G \succeq H$ implies $\text{term}(G) = \text{term}(H)$.

Proof  
Let $f : G \to H$ be the graph morphism mapping $\text{root}_G$ to $\text{root}_H$. We show by induction that for each node $v$ in $G$, $\text{term}_G(v) = \text{term}_H(f_V(v))$. Consider the unique edge $e$ with $\text{res}(e) = v$, and let $\text{arg}(e) = v_1 \ldots v_n$. Suppose that $\text{term}_G(v_i) = \text{term}_H(f_V(v_i))$ for $i = 1, \ldots, n$. Then $\text{term}_G(v) = \text{lab}_G(e)(\text{term}_G(v_1), \ldots, \text{term}_G(v_n)) = \text{term}_H(f_V(v))$.
Figure 5: Collapsing and copying

\[
\text{lab}_H(f_E(e))(\text{term}_H(f_V(v_1)), \ldots, \text{term}_H(f_V(v_n))) = \text{term}_H(\text{res}(f_E(e))) = \text{term}_H(f_V(\text{res}(e))) = \text{term}_H(f_V(v)).
\]

The induction principle just used allows to show that a property \( P \) holds for all nodes of a term graph. This principle is called \textit{bottom-up induction} and is as follows:

\textit{For all edges} \( e \), \textit{show that} \( P \) \textit{holds for} \( \text{res}(e) \) \textit{if} \( P \) \textit{holds for all nodes in} \( \text{arg}(e) \).

In the following, we will frequently use term graphs with minimal or maximal sharing.

\textbf{Definition 3.7 (Tree and fully collapsed term graph)}

A term graph \( G \) is a \textit{tree} if there is no \( H \) with \( G \prec H \), while \( G \) is \textit{fully collapsed} if there is no \( H \) with \( G \succ H \).

For example, the middle graph in Figure 5 is fully collapsed. The following characterization of trees and fully collapsed term graphs is easy to verify.

\textbf{Lemma 3.8}

(1) \textit{A term graph} \( G \) \textit{is a tree if and only if there is a unique path from root}_\( G \) \textit{to each other node}.

(2) \textit{A term graph} \( G \) \textit{is fully collapsed if and only if for all nodes} \( v \) \textit{and} \( w \), \( \text{term}_G(v) = \text{term}_G(w) \) \textit{implies} \( v = w \).

The next lemma is proved in [89].
Lemma 3.9
For every term graph $G$, there is a unique tree $\Delta G$ and a unique fully collapsed term graph $\nabla G$ such that

$$\Delta G \succeq G \succeq \nabla G.$$ 

Hence, $\Delta G$ and $\nabla G$ are the normal forms of $G$ with respect to $\prec$ and $\succ$. Note that by Lemma 3.6, $G$, $\Delta G$ and $\nabla G$ represent the same term.

Definition 3.10 (Bisimilarity)
Two term graphs $G$ and $H$ are bisimilar, denoted by $G \sim H$, if $\text{term}(G) = \text{term}(H)$.

The three graphs in Figure 5, for instance, are bisimilar (although the two outer graphs are related neither by collapsing nor by copying).

The notion of bisimilarity stems from the theory of concurrency (see for example [79]) and was adopted for term graphs in [4]. Given a term graph $G$, call the set $[G] = \{G' \mid G \sim G'\}$ the bisimilarity class of $G$. This class is partially ordered by $\succeq$.

Theorem 3.11 ([4])
For every term graph $G$, $([G], \succeq)$ is a complete lattice.

Clearly, $\Delta G$ and $\nabla G$ are the greatest and the least element in $[G]$, respectively. Thus, bisimilarity can be characterized as follows.

Corollary 3.12
For all term graphs $G$ and $H$, the following are equivalent:

1. $G \sim H$.
2. $\Delta G = \Delta H$.
3. $\nabla G = \nabla H$.

Given a term $t$, we write $\Delta t$ and $\nabla t$ for the unique tree and fully collapsed term graph representing $t$, respectively. Hence $\Delta G = \Delta \text{term}(G)$ and $\nabla G = \nabla \text{term}(G)$ for every term graph $G$.

3.3 Bibliographic notes

Term graphs as defined above are a special case of so-called jungles which were introduced in [42, 50]. Jungles are defined by requiring only conditions (2) and
(3) of Definition 3.1, so they can have several roots and need not be connected. Moreover, function symbols are equipped with a list of argument sorts and a result sort, and sorts are represented as node labels. For results about graph rewriting on jungles, we refer to [75, 43, 51, 61, 23, 89].

In the literature there exists a variety of definitions of term graphs. Besides hypergraphs, directed graphs, terms with labels, and recursion equations have been used as underlying structures.

Acyclic graphs have been dealt with in [34, 95, 96, 97], while [83, 92, 59, 15, 37, 63, 32] also consider cyclic graphs.

By equipping function symbols with additional labels, sharing of different occurrences of a subterm in a term can be expressed through identical labels. Such labelled terms correspond to acyclic term graphs and have been studied in [76, 74, 82].

In [36, 4, 2, 67], systems of recursion equations realize finite and infinite terms with sharing.

As to the complexity of collapsing, arbitrary term graphs can be made fully collapsed in time $O(n \log n)$, where $n$ is the size of term graphs. This bound reduces to $O(n)$ for term graphs over finite sets of function symbols and variables. See [30, 47, 38].
4 Term Graph Rewriting

In this section we define the transformation of term graphs by applications of term rewrite rules, introducing the notion of term graph rewriting. A fundamental property of this computational model is its soundness with respect to term rewriting. We also consider the addition of collapse and copy steps, and rewriting modulo bisimilarity. Collapsing sometimes speeds up the evaluation of term graphs considerably, which we show by an example.

4.1 Term rewriting

We first recall some basic concepts of term rewriting systems. For a comprehensive introduction, the reader may consult the textbook [11] or one of the surveys [54, 10, 28, 66, 84, 57].

Let $T_{\Sigma,X}$ be the set of all terms over $\Sigma$ and $X$. A substitution is a mapping $\sigma: T_{\Sigma,X} \rightarrow T_{\Sigma,X}$ such that $\sigma(c) = c$ for every constant $c$, and $\sigma(f(t_1,\ldots,t_n)) = f(\sigma(t_1),\ldots,\sigma(t_n))$ for every composite term $f(t_1,\ldots,t_n)$.

A term rewrite rule is a pair $(l,r)$ of terms, written $l \rightarrow r$, such that

1. $l$ is not a variable, and
2. all variables in $r$ occur also in $l$.

Such a rule is left-linear (resp. right-linear) if no variable occurs more than once in $l$ (resp. $r$). A term rewriting system $(\Sigma,\mathcal{R})$ consists of a set $\Sigma$ of function symbols and a set $\mathcal{R}$ of rules over $T_{\Sigma,X}$. A term rewriting system is left-linear (resp. right-linear) if all its rules are.

Definition 4.1 (Term rewriting)

The rewrite relation $\rightarrow$ on $T_{\Sigma,X}$ induced by a term rewriting system $\mathcal{R}$ is defined as follows: $t \rightarrow u$ if there is a rule $l \rightarrow r$ in $\mathcal{R}$ and a substitution $\sigma$ such that

1. $\sigma(l)$ is a subterm of $t$, and
2. $u$ is obtained from $t$ by replacing an occurrence of $\sigma(l)$ by $\sigma(r)$.

The following considerations aim at a fundamental result linking term rewriting with logic. It will show that term rewriting is a complete method for proving that an equation is a consequence of a given equational specification (having the form of a term rewriting system).

16
An algebra $A$ over $\Sigma$ consists of a non-empty set $D_A$, distinguished elements $c_A \in D_A$ for all constants $c$ in $\Sigma$, and $n$-ary functions $f_A: D_A^n \rightarrow D_A$ for all function symbols $f$ in $\Sigma$ with $\text{arity}(f) = n \geq 1$. An assignment (or valuation) is a mapping $v: X \rightarrow D_A$, which is extended to a mapping $v: T_{\Sigma,X} \rightarrow D_A$ by sending constants $c$ to $c_A$ and composite terms $f(t_1, \ldots, t_n)$ to $f_A(v(t_1), \ldots, v(t_n))$.

An equation is a pair $\langle l, r \rangle$ of terms, written $l \approx r$, and is valid in an algebra $A$ if $v(l) = v(r)$ for all assignments $v: X \rightarrow D_A$. In other words, $l \approx r$ stands for the formula $\forall x_1 \ldots \forall x_n l = r$ in predicate logic with equality, where $x_1, \ldots, x_n$ are the variables occurring in $l$ and $r$. An equational specification is a set of equations. A model of an equational specification $E$ is an algebra $A$ in which all equations of $E$ are valid. We write $E \models t \approx u$ if an equation $t \approx u$ is a consequence of $E$, that is, if it is valid in all models of $E$.

As every term rewriting system $\mathcal{R}$ is an equational specification (but not vice versa), we can speak of the models of $\mathcal{R}$ and of validity in these models. The following fundamental result is due to Birkhoff (see [11]).

**Theorem 4.2 (Completeness of term rewriting)**

For all terms $t$ and $u$,

$$\mathcal{R} \models t \approx u \text{ if and only if } t \xrightarrow{*} u.$$  

Thus, validity in the models of $\mathcal{R}$ coincides with convertibility by term rewriting. Although this is undecidable in general, the result provides a decision procedure for equational validity in the case where $\mathcal{R}$ is finite and has a confluent and terminating rewrite relation $\rightarrow$. In this case it suffices to rewrite $t$ and $u$ as long as possible, obtaining unique normal forms $t\downarrow$ and $u\downarrow$. Then $t \xrightarrow{*} u$ if and only if $t\downarrow = u\downarrow$.

### 4.2 Term graph rewriting

In this subsection we define the application of term rewrite rules to term graphs. Given a rule $l \rightarrow r$ and a term graph $G$, first one has to find the left-hand side $l$ in $G$. Technically, this amounts to find a graph morphism into $G$ starting from a graph representation of $l$ in which only repeated variables are shared.

**Definition 4.3 ($\check{t}$ and $\Diamond t$)**

For every term $t$, let $\check{t}$ be the term graph representing $t$ such that only variables are shared. That is, there is a graph morphism $f: \Delta t \rightarrow \check{t}$ such that for all distinct edges $e_1$ and $e_2$,

$$f_E(e_1) = f_E(e_2) \text{ if and only if } \text{lab}_E(e_1) = \text{lab}_E(e_2) \in X.$$
The graph resulting from $\Diamond t$ after removing all edges labelled with variables is denoted by $\Diamond t$.

For example, Figure 6 shows the graphs $\Delta f(x,x)$, $\Diamond f(x,x)$ and $\Diamond f(x,x)$. Note that the latter graph is not a term graph according to Definition 3.1. It may be regarded as a term graph with an "open node". (In [15, 89], such graphs are also regarded as term graphs, and term graphs without open nodes are said to be closed.)

Figure 6: The graphs $\Delta f(x,x)$, $\Diamond f(x,x)$ and $\Diamond f(x,x)$

For each node $v$ in a term graph $G$, we denote by $G[v]$ the (standard) term graph isomorphic to the subgraph of $G$ consisting of all nodes reachable from $v$ and all edges having these nodes as result nodes.

**Definition 4.4 (Instance and redex)**
A term graph $L$ is an instance of a term $I$ if there is graph morphism $\Diamond I \rightarrow L$ sending root$\Diamond I$ to root$L$. Given a node $v$ in a term graph $G$ and a term rewrite rule $l \rightarrow r$, the pair $(v, l \rightarrow r)$ is a redex if $G[v]$ is an instance of $l$.

We will also call the subgraph $G[v]$ a redex if there is no ambiguity or if the applied rule is irrelevant.

**Definition 4.5 (Term graph rewriting)**
Let $G$ be a term graph containing a redex $(v, l \rightarrow r)$. Then there is a proper rewrite step $G \Rightarrow_{e, l \rightarrow r} H$, where $H$ is the term graph constructed as follows:

1. $G_1 = G - \{e\}$ is the graph obtained from $G$ by removing the unique edge $e$ satisfying $\text{res}(e) = v$.

2. $G_2$ is the graph obtained from the disjoint union $G_1 + \Diamond r$ by
   - identifying $v$ with root$\Diamond r$,
   - identifying the image of $\text{res}(e_1)$ with $\text{res}(e_2)$, for each pair $(e_1, e_2) \in E_{\Diamond l} \times E_{\Diamond r}$ with $\text{lab}_{\Diamond l}(e_1) = \text{lab}_{\Diamond r}(e_2) \in X$. 

18
We denote such a rewrite step also by $G \Rightarrow_h H$ or simply by $G \Rightarrow H$.

**Example 4.1**
Figure 7 shows the three intermediate steps in the construction of a term graph rewrite step. The term rewrite rule applied to $G$ is $x \times (y + z) \rightarrow (x \times y) + (x \times z)$. In $G$ and $G_3$, shaded nodes and edges belong to the occurrences of $\Diamond x \times (y + z)$ and $\Diamond (x \times y) + (x \times z)$, respectively. Note that the variables $y$ and $z$ correspond to the same node, that is, the graph morphism $\Diamond x \times (y + z) \rightarrow G$ identifies the nodes representing $y$ and $z$.

![Figure 7: The intermediate steps in the construction of a term graph rewrite step](image)

The term graph rewrite relation $\Rightarrow$ is sound with respect to term rewriting in the sense that every proper step $G \Rightarrow_{\nu, l \rightarrow r} H$ corresponds to a sequence of applications (or a parallel application) of $l \rightarrow r$ to several occurrences of the subterm $\operatorname{term}_G(v)$ in $\operatorname{term}(G)$. This explains the possible gain in time efficiency when passing from term rewriting to term graph rewriting.

**Theorem 4.6 (Soundness [50])**
For all term graphs $G$ and $H$,

$$G \Rightarrow H \text{ implies } \operatorname{term}(G) \xrightarrow{\dagger} \operatorname{term}(H).$$

---

1More precisely, $H$ is the unique standard term graph isomorphic to the term graph constructed in this step.
More precisely, we have \( \text{term}(G) \rightarrow^n \text{term}(H) \), where \( n \) is the number of paths from root \( G \) to \( v \).

A proof of this result can be found in [51, 89].

**Example 4.2**
Consider the application of the rule \( 0 + x \rightarrow x \) shown in Figure 8. (Note that the graph \( \hat{x} \) is a single node which is identified with the root of the redex and with the result node of the edge labelled with \( s \).) There are three paths from the root of the left graph to the root of the redex, and the term graph rewrite step corresponds to the threefold term rewrite step
\[
((0 + s(0)) \times (0 + s(0))) + (0 + s(0)) \rightarrow^3 (s(0) \times s(0)) + s(0).
\]  

![Figure 8: An application of the rule 0 + x \( \rightarrow x \)](image)

4.3 Incorporating collapsing and copying

In the next section we will see that not all term rewriting derivations can be simulated by \( \Rightarrow \)-derivations. This incompleteness can partly be overcome by allowing proper collapse or copy steps besides applications of term rewrite rules. Completeness with respect to term rewriting can be achieved by adding both collapsing and copying to \( \Rightarrow \), or by using rewriting modulo bisimilarity. This topic will be discussed in the next section. In this subsection we define the mentioned extensions, present an example in which collapsing speeds up the evaluation of term graphs, and relate \( \Rightarrow \) to rewriting with collapsing.

**Definition 4.7** (\( \Rightarrow_{\text{coll}}, \Rightarrow_{\text{copy}} \) and \( \Rightarrow_{\text{bi}} \))
The relations \( \Rightarrow_{\text{coll}}, \Rightarrow_{\text{copy}} \) and \( \Rightarrow_{\text{bi}} \) on term graphs are defined as follows:
\[
\begin{align*}
\Rightarrow_{\text{coll}} &= \Rightarrow \cup \succ, \\
\Rightarrow_{\text{copy}} &= \Rightarrow \cup \prec, \\
\Rightarrow_{\text{bi}} &= \Rightarrow \cup \succ \cup \prec.
\end{align*}
\]
We refer to $\Rightarrow$, $\Rightarrow_{\text{coll}}$, $\Rightarrow_{\text{copy}}$ and $\Rightarrow_{\text{bi}}$ as plain term graph rewriting, term graph rewriting with collapsing, term graph rewriting with copying, and term graph rewriting with collapsing and copying, respectively.

The relations $\Rightarrow_{\text{coll}}$, $\Rightarrow_{\text{copy}}$ and $\Rightarrow_{\text{bi}}$ are sound in the sense of Theorem 4.6 if we replace $\rightarrow^+$ by $\rightarrow^*$. For, by Lemma 3.6, collapse and copy steps do not change the term represented by a term graph. Note also that $\Rightarrow_{\text{bi}}$ contains bisimilarity since $G \sim H$ implies $G \triangleq G \succeq H$ (see Lemma 3.9 and Corollary 3.12).

**Example 4.3**

In certain cases, collapsing can speed up evaluation processes drastically. A prime example is the specification of the Fibonacci function:

\[
\begin{align*}
\text{fib}(0) & \Rightarrow 0 \\
\text{fib}(\text{s}(0)) & \Rightarrow \text{s}(0) \\
\text{fib}(\text{s}(\text{s}(x))) & \Rightarrow \text{fib}(\text{s}(x)) + \text{fib}(x)
\end{align*}
\]

Using these three rules, evaluating a term of the form $\text{fib}(\text{s}^n(0))$ by term rewriting requires a number of rewrite steps exponential in $n$ (see [1]). One easily observes that the same number of steps is needed for plain term graph rewriting. After replacing $\Rightarrow$ by $\Rightarrow_{\text{coll}}$, however, it is possible to evaluate $\text{fib}(\text{s}^n(0))$ in a linear number of steps. The evaluation strategy can be described as follows: (1) Collapse steps have priority over proper rewrite steps and produce fully collapsed term graphs. (2) Out of two $\text{fib}$-redexes, the one representing the greater number is reduced. See Figure 9 for an illustration of this strategy. It is not difficult to verify that, for $n \geq 2$, this procedure evaluates $\text{fib}(\text{s}^n(0))$ in $2n + 1$ steps (viz. $n + 1$ proper rewrite steps and $n$ collapse steps).

The next section will show that apart from speeding up evaluation, collapsing is necessary to cope with non-left-linear rewrite rules. If no such rules are present, $\Rightarrow_{\text{coll}}$ can be simulated by $\Rightarrow$ as follows.

**Theorem 4.8**

If $\mathcal{R}$ is left-linear, then for all term graphs $G$ and $H$,

\[
G \xRightarrow{\text{coll}}^* H \text{ implies } G \xRightarrow{\text{coll}}^* H' \succeq H
\]

for some term graph $H'$.

Theorem 4.8 is a corollary of a result in [45] showing that every $\Rightarrow_{\text{coll}}$-derivation can be transformed into a so-called minimally collapsing derivation.

We conclude this subsection by introducing rewriting modulo bisimilarity, where collapsing and copying are “built in” in the sense that rewrite steps transform bisimilarity classes rather than term graphs.
Figure 9: Collapsing to speed up evaluation
Definition 4.9 (Rewriting modulo bisimilarity)

The relation $\Rightarrow_\sim$ on bisimilarity classes is defined as follows: $[G] \Rightarrow_\sim [H]$ if there are term graphs $G'$ and $H'$ such that $G \sim G' \Rightarrow H' \sim H$. We refer to $\Rightarrow_\sim$ as term graph rewriting modulo bisimilarity.

For example, in Figure 9 there exists a rewrite step between the bisimilarity classes of the second and the fourth term graph.

Term graph rewriting modulo bisimilarity generalizes term rewriting in that for all term graphs $G$ and $H$, $\text{term}(G) \rightarrow \text{term}(H)$ implies $[G] \Rightarrow_\sim [H]$, and $[G] \Rightarrow_\sim [H]$ implies $\text{term}(G) \rightarrow^+ \text{term}(H)$.

4.4 Bibliographic notes

Term graph rewriting was first studied in [95], where it was shown that non-overlapping rules give rise to a subcommutative rewrite relation. The name term graph rewriting was introduced in [15]. This paper focuses on normalizing strategies and states the soundness of $\Rightarrow$ for left-linear rules.

In [15, 63], term graph rewrite rules are considered which operate on possibly cyclic term graphs. The application of such a rule involves the redirection of all edges pointing to the root of the left-hand side, to the root of the right-hand side. (An alternative, “transitive” version of redirection is investigated in [14].)

The approach of [15, 63] is extended in [37] by allowing to choose in a rewrite step among several structure sharing schemes which perform a certain collapsing or copying. Soundness with respect to a certain kind of infinite term rewriting is shown for left-linear, left-finite, left-acyclic term graph rewrite rules.

In [43, 51], jungles (see subsection 3.3) are evaluated by rules conforming to the double-pushout approach to graph rewriting [33, 21]. The evaluation rules are obtained by translating term rewrite rules, and their application corresponds to the effect of steps (1) and (2) in Definition 4.5—so there is no garbage collection. Collapse steps (called folding steps in [43, 51, 89]) are also specified by suitable graph rewrite rules. The relation between jungle evaluation and the present setting is discussed in [89].

Similar to jungle evaluation, in [23] rewrite steps on jungles are defined by two pushouts. The difference is that one considers the category of jungles instead of the category of hypergraphs. This implicitly enforces a kind of minimal collapsing in evaluation steps with non-left-linear term rewrite rules.

A categorical treatment of garbage collection is given in [12]. In [19], a description of term graph rewriting by a 2-category is presented.
5 Completeness

In this section we consider the completeness of term graph rewriting for simulating arbitrary term rewrite derivations (Subsection 5.1) and for computing term normal forms (Subsection 5.2). We will see that in general, \( \Rightarrow_{\text{coll}} \) and \( \Rightarrow_{\text{copy}} \) are incomplete in these respects. Nevertheless, both relations are complete—in the same sense as term rewriting is—for proving validity of equations, and can compute term normal forms over certain subclasses of term rewriting systems.

5.1 Simulating arbitrary term rewrite derivations

From Theorem 4.6 we know that for every term graph rewrite derivation \( G \Rightarrow^* H \) there is a corresponding term rewrite derivation \( \text{term}(G) \Rightarrow^* \text{term}(H) \). The next two examples will show that the converse does not hold, even if we extend \( \Rightarrow \) to \( \Rightarrow_{\text{coll}} \) or \( \Rightarrow_{\text{copy}} \).

Example 5.1

One obstacle to the completeness of plain term graph rewriting are non-left-linear term rewrite rules. For instance, the rule \( \text{eq}(x,x) \rightarrow \text{true} \) cannot be applied to the tree \( \Delta \text{eq}(0,0) \) because there is no graph morphism \( \Delta \text{eq}(x,x) \rightarrow \Delta \text{eq}(0,0) \) (see Figure 10). Hence, \( \Delta \text{eq}(0,0) \) is not reducible by \( \Rightarrow \) or \( \Rightarrow_{\text{copy}} \) although the represented term is reducible. Figure 10 also shows how to overcome the problem by collapsing: identifying the two occurrences of 0 enables a subsequent application of the rewrite rule.

Figure 10: Collapsing to enable a rule application
Example 5.2
Even for left-linear systems, certain term rewrite derivations do not correspond to derivations by \( \Rightarrow \) or \( \Rightarrow_{\text{coll}} \). Consider, for example, the rules \( f(x) \rightarrow g(x, x) \) and \( a \rightarrow b \). The derivation \( f(a) \rightarrow g(a, a) \rightarrow g(a, b) \) cannot be simulated by \( \Rightarrow \) or \( \Rightarrow_{\text{coll}} \) because the application of the first rule leads to a shared constant \( a \) (see Figure 11). But this time we can simulate the given term rewrite derivation by

\[
\begin{array}{c}
  f \\
  \downarrow \\
  a \\
\end{array} 
\Rightarrow 
\begin{array}{c}
  g \\
  \downarrow \\
  a \\
\end{array} 
\Rightarrow 
\begin{array}{c}
  g \\
  \downarrow \\
  b \\
\end{array}
\]

Figure 11: Applications of the rules \( f(x) \rightarrow g(x, x) \) and \( a \rightarrow b \)

\( \Rightarrow_{\text{copy}} \), as shown in Figure 12.

\[
\begin{array}{c}
  f \\
  \downarrow \\
  a \\
\end{array} 
\Rightarrow 
\begin{array}{c}
  g \\
  \quad \quad \quad \quad \quad \quad \quad \quad \\
  \quad \quad \quad \quad \quad \quad \quad \quad \\
  a \\
\end{array} 
\Rightarrow 
\begin{array}{c}
  g \\
  \quad \quad \quad \quad \quad \quad \quad \quad \\
  a \\
\end{array}
\]

Figure 12: A \( \Rightarrow_{\text{copy}} \)-derivation

The above examples show that in general both collapsing and copying are needed to simulate term rewrite derivations. This is reflected by the following lemma.

Lemma 5.1 (Simulation of term rewrite steps [88])
For every term rewrite step \( t \rightarrow u \) there are term graphs \( T \) and \( U \) such that

\[ \Delta t \succeq T \Rightarrow U \preceq \Delta u. \]

To illustrate Lemma 5.1, consider the rule \( f(x + x) \rightarrow f(x) + f(x) \) which is neither left-linear nor right-linear. It admits the following term rewrite step:

\[ g(f(0 + 0), 0) \rightarrow g(f(0) + f(0), 0) \]

Figure 13 shows how to simulate this step by term graph rewriting. In general, the subtree of \( \Delta t \) corresponding to the replaced subterm \( \sigma(t) \) is compressed as much as is necessary to apply the term rewrite rule. The resulting graph contains only one path from the root to the redex, and hence the application of the rule simulates a single term rewrite step.

Using Lemma 5.1, it is straightforward to show that every sequence of term rewrite steps can be simulated if both collapsing and copying are present.
Theorem 5.2 (Completeness of \( \Rightarrow_{bi} \) and \( \Rightarrow_{\sim} \) [7, 6])

For all term graphs \( G \) and \( H \), the following are equivalent:

1. \( \text{term}(G) \rightarrow^* \text{term}(H) \).
2. \( G \Rightarrow_{bi}^* H \).
3. \( [G] \Rightarrow_{\sim}^* [H] \).

Proof

By the definitions of \( \Rightarrow_{bi} \) and \( \Rightarrow_{\sim} \) it is clear that (2) implies (3), and (3) implies (1) by soundness of \( \Rightarrow \). So it remains to show that (1) implies (2). By Lemma 5.1, for every term rewrite step \( t \rightarrow u \) there is a derivation \( \Delta t \Rightarrow_{bi}^+ \Delta u \). Hence, by induction on the length of derivations, \( \text{term}(G) \rightarrow^* \text{term}(H) \) implies \( \Delta \text{term}(G) \Rightarrow_{bi}^+ \Delta \text{term}(H) \). Since \( G \leq \Delta \text{term}(G) \) and \( \Delta \text{term}(H) \geq H \), it follows \( G \Rightarrow_{bi}^* H \). 

The equivalence of (1) and (3) remains valid if we replace \( \rightarrow^* \) and \( \Rightarrow_{\sim}^* \) by \( \rightarrow^+ \) and \( \Rightarrow_{\sim}^+ \), respectively. In contrast, if \( G \Rightarrow_{bi} H \) is a collapse or copy step, then neither \( \text{term}(G) \rightarrow^+ \text{term}(H) \) nor \( [G] \Rightarrow_{\sim}^+ [H] \) will hold in general.

Combining the completeness of \( \Rightarrow_{bi} \) with the observation that \( \Leftrightarrow_{bi} = \Leftrightarrow_{\text{coll}} = \Leftrightarrow_{\text{copy}} \), we obtain the following corollary of Theorem 5.2.

Corollary 5.3 (Completeness of \( \Rightarrow_{\text{coll}} \) and \( \Rightarrow_{\text{copy}} \))

For all term graphs \( G \) and \( H \), the following are equivalent:

1. \( \text{term}(G) \leftrightarrow^* \text{term}(H) \).
2. \( G \leftrightarrow_{\text{coll}}^* H \).
3. \( G \leftrightarrow_{\text{copy}}^* H \).
Thus, an equation \( t \approx u \) is valid in the models of \( \mathcal{R} \) if and only if there is a sequence of \( \Leftarrow_{\text{coll}} \) respectively \( \Leftarrow_{\text{copy}} \)-steps between two term graphs representing \( t \) and \( u \). In other words, term graph rewriting with collapsing or copying is complete for proving equational validity in the same sense as term rewriting is (cf. Theorem 4.2). Moreover, the decision procedure for equational validity described below Theorem 4.2 can be replaced by a corresponding procedure using \( \Rightarrow_{\text{coll}} \) or \( \Rightarrow_{\text{copy}} \). If, say, \( \Rightarrow_{\text{coll}} \) is convergent, one represents the terms of an equation by term graphs and reduces these to normal forms by \( \Rightarrow_{\text{coll}} \). The equation is valid if and only if the resulting normal forms are equal.

Note that plain term graph rewriting lacks this kind of completeness: in Example 5.1 there is no conversion \( \Delta\text{eq}(0, 0) \Leftarrow^* \Delta\text{true} \) although \( \text{eq}(0, 0) \approx \text{true} \) is a valid equation, and in Example 5.2 there does not exist a conversion \( \Delta f(a) \Leftarrow^* \Delta g(a, b) \).

5.2 Graph-reducibility

In the previous subsection we saw that \( \Rightarrow, \Rightarrow_{\text{coll}} \) and \( \Rightarrow_{\text{copy}} \) are not able to simulate arbitrary term rewrite derivations. We now relax the requirement and consider only derivations ending in normal forms.

**Definition 5.4 (Graph-reducibility)**

A term rewriting system \( \mathcal{R} \) is **graph-reducible** by a binary relation \( \Rightarrow \) on term graphs if the following holds for every term graph \( G \):

1. For every normal form \( H \) of \( G \) with respect to \( \Rightarrow \), \( \text{term}(H) \) is a normal form of \( \text{term}(G) \) with respect to \( \rightarrow \).
2. If \( \text{term}(G) \) has a normal form with respect to \( \rightarrow \), then \( G \) has a normal form with respect to \( \Rightarrow \).

The system \( \mathcal{R} \) is **strongly graph-reducible** by \( \Rightarrow \) if it satisfies (1) and if for every term graph \( G \) and every normal form \( t \) of \( \text{term}(G) \), \( G \) has a normal form representing \( t \).

Condition (1) ensures soundness of \( \Rightarrow \) in the sense that every derivation ending in a normal form computes a term normal form. Condition (2) expresses completeness: a term graph has a normal form whenever its represented term has.

From Example 5.1 we already know that non-left-linear systems are not graph-reducible by \( \Rightarrow \) and \( \Rightarrow_{\text{copy}} \) in general: for \( \mathcal{R} = \{ \text{eq}(x, x) \rightarrow \text{true} \} \), the tree \( \Delta\text{eq}(0, 0) \) is a normal form with respect to \( \Rightarrow \) and \( \Rightarrow_{\text{copy}} \) although \( \text{eq}(0, 0) \) is reducible by \( \mathcal{R} \).
On the other hand, we will see that the system \( \{ f(x) \rightarrow g(x, x), a \rightarrow b \} \) of Example 5.2 is graph-reducible by \( \Rightarrow \), \( \Rightarrow_{\text{coll}} \) and \( \Rightarrow_{\text{copy}} \).

**Definition 5.5 (Non-overlapping and orthogonal systems)**

A term \( s \) overlaps a term \( t \) in a subterm \( u \) of \( t \) if \( u \) is not a variable and if there are substitutions \( \sigma \) and \( \tau \) such that \( \sigma(s) = \tau(u) \). A term rewriting system \( \mathcal{R} \) is non-overlapping if for all rules \( l_1 \rightarrow r_1 \) and \( l_2 \rightarrow r_2 \) in \( \mathcal{R} \), \( l_1 \) overlaps \( l_2 \) in a subterm \( u \) only if \( u = l_2 \) and \( (l_1 \rightarrow r_1) = (l_2 \rightarrow r_2) \). If \( \mathcal{R} \) is non-overlapping and additionally left-linear, then it is an orthogonal system.

**Theorem 5.6 ([15])**

Every orthogonal term rewriting system is graph-reducible by \( \Rightarrow \).

In fact, orthogonal systems are strongly graph-reducible since it is well-known that every term has at most one normal form with respect to \( \Rightarrow \).

To see that left-linearity alone is not sufficient for graph-reducibility by \( \Rightarrow \), extend the system \( \{ f(x) \rightarrow g(x, x), a \rightarrow b \} \) by the rules \( g(a, b) \rightarrow c \) and \( g(b, b) \rightarrow f(a) \). Then one can easily check that \( \Delta f(a) \) does not have a normal form while \( f(a) \) reduces to the term normal form \( c \).

The next two results establish graph-reducibility by \( \Rightarrow_{\text{coll}} \) for classes of systems that need neither be left-linear nor non-overlapping. Instead, graph-reducibility holds when certain restricted forms of term rewriting derivations suffice to normalize terms.

Define the parallel rewrite relation \( \Rightarrow \) on \( T_{\Sigma, X} \) by modifying clause (2) of Definition 4.1 as follows: “\( u \) is obtained from \( t \) by replacing all occurrences of \( \sigma(l) \) by \( \sigma(r) \).” Call a term rewriting system parallelly normalizing if for every term having a normal form, there is a normal form \( u \) such that \( t \Rightarrow^* u \). The class of parallelly normalizing systems includes, for example, all orthogonal and all terminating term rewriting systems.

**Theorem 5.7 ([86])**

Every parallelly normalizing term rewriting system is graph-reducible by \( \Rightarrow_{\text{coll}} \).

A term rewrite step \( t \rightarrow u \) is an innermost step if all proper subterms of the replaced subterm \( \sigma(l) \) are normal forms. A term rewriting system is innermost normalizing if every term can be rewritten to a normal form by a sequence of innermost rewrite steps. The classes of innermost normalizing and parallelly normalizing term rewriting systems are incomparable (see [69]).

**Theorem 5.8 ([69])**

Every innermost normalizing term rewriting system is graph-reducible by \( \Rightarrow_{\text{coll}} \).
We conclude this subsection by considering graph-reducibility by $\Rightarrow_{\text{copy}}$. The result below follows from the fact that if all term rewrite rules are left-linear, then for every term rewrite derivation $t \rightarrow^* u$ there is a term graph rewrite derivation $\Delta t \Rightarrow^*_{\text{copy}} \Delta u$ (see [6]).

**Theorem 5.9**
Every left-linear term rewriting system is strongly graph-reducible by $\Rightarrow_{\text{copy}}$.

### 5.3 Bibliographic notes

Completeness of $\Rightarrow_{\text{coll}}$ for proving equational validity was shown in [88]. Graph-reducibility was first considered in [15], where the lifting of certain term rewrite strategies to the setting of term graph rewriting is studied. A stronger notion than graph-reducibility is *adequacy*, which is treated in [63]. The definition of adequacy is tailored to orthogonal systems as it requires that every term rewrite sequence can be extended to a sequence that corresponds to some term graph rewrite sequence. In [63] it is shown that $\Rightarrow$ is adequate for orthogonal systems, and that orthogonal term graph rewriting with possibly cyclic graphs is adequate for a certain kind of infinitary, orthogonal term rewriting.
6 Termination

For several reasons, termination is an important property of rewriting systems. If a rewrite relation on term graphs is known to be terminating, every term graph can be reduced to a normal form simply by performing arbitrary rewrite steps as long as possible. Moreover, several properties that are generally undecidable become decidable in the presence of termination. For example, the transitive closure $\Rightarrow^*$ and the question whether $\Rightarrow$ is confluent are decidable then (provided $\mathcal{R}$ is finite). If $\Rightarrow$ is both terminating and confluent, it even gives rise to a decision procedure for equational validity in the models of $\mathcal{R}$ (see the remark below Corollary 5.3).

6.1 The relation to term rewriting

First we compare termination of term and term graph rewriting. We will see that the class of terminating term rewriting systems is properly included in the class of systems for which $\Rightarrow$ is terminating. By restricting attention to right-linear systems, however, termination of $\Rightarrow$ becomes equivalent to termination of $\Rightarrow$. As a consequence, undecidability of termination carries over from term rewriting to term graph rewriting.

The following theorem is a consequence of the soundness of $\Rightarrow$ and the fact that $\Rightarrow$ and $\Rightarrow^*$ are terminating relations.

**Theorem 6.1**

*If $\Rightarrow$ is terminating, then $\Rightarrow$, $\Rightarrow_{coll}$, $\Rightarrow_{copy}$ and $\Rightarrow_\sim$ are terminating as well.*

Note that this result does not hold for $\Rightarrow_{bi}$, since if both collapsing and copying are present, there may be an infinite sequence of alternating collapse and copy steps. For $\Rightarrow_\sim$, the reverse of Theorem 6.1 also holds [6], while for $\Rightarrow_{copy}$ one has to require that $\mathcal{R}$ is left-linear (otherwise the system $\{f(x, x) \rightarrow f(a, a)\}$ is a counterexample).

It is worth noting that by Theorem 6.1, the wide range of techniques for proving termination of term rewriting (see for example [11, 27]) can be used to prove termination of term graph rewriting. However, the next example demonstrates that term graph rewriting in form of $\Rightarrow$ and $\Rightarrow_{coll}$ terminates “more often” than term rewriting.
**Example 6.1**
Consider the following two rules:

\[
\begin{align*}
  f(a, b, x) & \rightarrow f(x, x, x) \\
  a & \rightarrow b
\end{align*}
\]

Term rewriting is not terminating as there is an infinite rewrite sequence:

\[
f(a, b, a) \rightarrow f(a, a, a) \rightarrow f(a, b, a) \rightarrow \ldots
\]

In contrast, \( \Rightarrow \) and \( \Rightarrow_{\text{coll}} \) are terminating. This can be proved by means of the following function \( \tau \) from term graphs to natural numbers. For every term graph \( G \), define \( \tau(G) = m + n + p \), where \( m \) is the number of \( f \)-labelled edges the first two argument nodes of which are distinct, \( n \) is the number of \( a \)-labelled edges, and \( p \) is the number of nodes in \( G \). It is not difficult to check that for every step \( G \Rightarrow_{\text{coll}} H \), we have \( \tau(G) > \tau(H) \). Thus, every sequence of \( \Rightarrow_{\text{coll}} \)-steps (and hence every sequence of \( \Rightarrow \)-steps) must eventually terminate.

For the rest of this section, we concentrate on term graph rewriting with collapsing. We just remark that by the proof of Theorem 4.8, if \( \mathcal{R} \) is left-linear, then \( \Rightarrow \) is terminating if and only if \( \Rightarrow_{\text{coll}} \) is terminating.

**Theorem 6.2**
If \( \mathcal{R} \) is right-linear, then \( \Rightarrow_{\text{coll}} \) is terminating if and only if \( \Rightarrow \) is terminating.

**Proof**
The “if”-direction is contained in Theorem 6.1. The “only if”-direction follows from the proof of Lemma 5.1. There, the collapsing \( \Delta t \supset T \) is chosen such that the garbage collection phase of the rewrite step \( T \Rightarrow U \) removes each edge that has in \( T \) a shared argument node (meaning that this node is also an argument node of some other edge or appears more than once in the argument string). Moreover, by right-linearity, the inserted graph \( R_r \) is a tree (where \( r \) is the right-hand side of the applied rewrite rule). It follows \( U = \Delta u \). Hence every term rewrite sequence can be simulated by a sequence of \( \Rightarrow_{\text{coll}} \)-steps, which implies the proposition. \( \square \)

**Corollary 6.3**
The following problem is undecidable in general:

Instance: A finite term rewriting system \( \mathcal{R} \).
Question: Is \( \Rightarrow_{\text{coll}} \) terminating?

**Proof**
It is known that it is undecidable in general whether a finite, right-linear term rewriting system is terminating or not (see [53, 11]). Hence, by Theorem 6.2, termination of \( \Rightarrow_{\text{coll}} \) cannot be decidable either. \( \square \)
6.2 Combined systems

Proving termination of term or term graph rewriting is a difficult task which is unsolvable in general. A desirable method for handling a possibly large system $\mathcal{R}$ is decomposing it into subsystems and proving termination separately for these. To make this approach work, though, one needs criteria ensuring that the union of two terminating systems is again terminating. That the latter may fail can be seen by putting together the terminating systems $\{a \rightarrow b\}$ and $\{b \rightarrow a\}$, yielding a non-terminating system. Even worse, Toyama [98] showed that the disjoint union of two terminating term rewriting systems need not be terminating. He gave the following counterexample.

**Example 6.2**

The two systems

$$\mathcal{R}_0 \{ \begin{align*}
f(0, 1, x) &\rightarrow f(x, x, x) \\
g(x, y) &\rightarrow x \\
g(x, y) &\rightarrow y
\end{align*} \}$$

have disjoint function symbols and are both terminating. But their union admits the following infinite rewrite sequence:

$$f(g(0, 1), g(0, 1), g(0, 1)) \xrightarrow{2} f(0, 1, g(0, 1)) \rightarrow f(g(0, 1), g(0, 1), g(0, 1)) \rightarrow \ldots$$

Toyama’s observation stimulated several researchers to establish sufficient conditions under which the disjoint union of term rewriting systems preserves termination (see [41, 81] and the references given there). The interesting fact, now, is that such conditions are not needed in the case of term graph rewriting. For, termination of $\Rightarrow_{\text{coll}}$ does behave modular with respect to disjoint unions. To demonstrate this by Toyama’s example, let us try to simulate by $\Rightarrow$ the infinite term rewrite sequence shown above. Starting with the tree $\Delta f(g(0, 1), g(0, 1), g(0, 1))$, one obtains the terminating derivations shown in Figure 14.

![Figure 14: Two terminating derivations](image-url)
The crucial point here is the application of the rule \( f(0,1, x) \rightarrow f(x, x, x) \). In term rewriting, this rule produces three copies of the subterm \( g(0, 1) \) which can be evaluated independently. In contrast, term graph rewriting yields a shared occurrence of this subterm, preventing that \( f(0, 1, x) \rightarrow f(x, x, x) \) can be applied again.

We will see that termination of \( \Rightarrow_{\text{coll}} \) for a composed system \( R_0 \cup R_1 \) can be guaranteed even when function symbols are shared between the left-hand sides respectively right-hand sides of \( R_0 \) and \( R_1 \).

**Definition 6.4 (Crosswise disjointness)**

Two term rewriting systems \( R_0 \) and \( R_1 \) are *crosswise disjoint* if the function symbols in the left-hand sides of the rules in \( R_i \) do not occur in the right-hand sides of the rules in \( R_{1-i} \), for \( i = 0, 1 \).

For example, the following systems are crosswise disjoint:

\[
R_0 = \begin{cases} 
  f(x) & \rightarrow g(x, x) \\
  a & \rightarrow b 
\end{cases} \\
R_1 = \begin{cases} 
  f(f(x)) & \rightarrow g(x, b) \\
  h(a, x) & \rightarrow h(b, x) 
\end{cases}
\]

In the following, we write \( \Rightarrow_R \) for the relation \( \Rightarrow_{\text{coll}} \) over a term rewriting system \( R \).

**Theorem 6.5 ([87])**

Let \( R_0 \cup R_1 \) be the union of two crosswise disjoint term rewriting systems. Then \( \Rightarrow_{R_0 \cup R_1} \) is terminating if and only if \( \Rightarrow_{R_0} \) and \( \Rightarrow_{R_1} \) are terminating.

The main motivation for this result is to facilitate termination proofs. But we can also use it to sharpen Corollary 6.3, obtaining a stronger undecidability result for termination.

**Corollary 6.6**

The following problem is undecidable in general:

**Instance:** A finite term rewriting system \( R \) such that \( \rightarrow \) is not terminating.

**Question:** Is \( \Rightarrow_R \) terminating?

**Proof**

If the above problem were decidable, we could decide termination of \( \Rightarrow_R \) for arbitrary finite systems \( R \) as follows. First construct the disjoint union \( R' = R + \{ f(a, b, x) \rightarrow f(x, x, x), \ a \rightarrow b \} \). By Example 6.1 and Theorem 6.5, \( R' \) is a non-terminating term rewriting system, and \( \Rightarrow_R \) is terminating if and only if
⇒_{R'} is terminating. Thus, if the above problem were decidable, we could decide whether ⇒_R is terminating or not. But this contradicts Corollary 6.3.

Coming back to the question when a composed system inherits termination from its components, we now consider an alternative to crosswise disjointness. The condition is based on partitioning the set of function symbols into defined symbols and constructors, where the former are the leftmost symbols in the left-hand sides of rules, and the latter are the remaining symbols.

**Definition 6.7 (Constructor-sharing)**
Two term rewriting systems \( R_0 \) and \( R_1 \) are constructor-sharing if the defined symbols of \( R_1 \) do not occur in \( R_{1-i} \), for \( i = 0, 1 \).

The proof of the following result was given in the framework of so-called non-copying term rewriting, but holds in the present setting as well.

**Theorem 6.8 ([74])**
Let \( R_0 \cup R_1 \) be the union of two constructor-sharing term rewriting systems. Then \( \Rightarrow_{R_0 \cup R_1} \) is terminating if and only if \( \Rightarrow_{R_0} \) and \( \Rightarrow_{R_1} \) are terminating.

An extended version of this result, also given in [74], additionally allows \( R_0 \) and \( R_1 \) to share defined symbols that do not occur in any right-hand side. In this form the result generalizes Theorem 6.5. An even more general result is presented in [71, 72], but we refrain from stating it because of its technically involved premise. In [72] one can also find a condition—more general then disjointness—guaranteeing that normalization of \( \Rightarrow_{\text{coll}} \) is preserved by combinations of systems.

### 6.3 A recursive path order on term graphs

In the two previous subsections we have seen examples of non-terminating term rewriting systems for which \( \Rightarrow_{\text{coll}} \) is terminating. This raises the question for termination proof techniques covering such systems. In this subsection we introduce a recursive path order on term graphs by analogy with the well-known order on terms [26, 27], and demonstrate its use for proving termination of \( \Rightarrow_{\text{coll}} \). Our exposition is based on [91], where a class of simplification orders on term graphs is established by extending Kruskal’s Tree Theorem [73] from trees to term graphs.

**Definition 6.9 (Top and immediate subgraphs)**
Let \( G \) be a term graph and \( e \) be the unique edge such that \( \text{att}_G(e) = \text{root}_G v_1 \ldots v_n \) for some nodes \( v_1, \ldots, v_n \). Then the top of \( G \), denoted by \( \text{top}_G \), is the subgraph consisting of \( e \) and the nodes \( \text{root}_G, v_1, \ldots, v_n \). The term graphs
\[G[v_1], \ldots, G[v_n]\] are the immediate subgraphs of \(G\). We denote by \(\text{Sub}_G\) the multiset \(\{G[v_1], \ldots, G[v_n]\}\).

Recall that a preorder is a reflexive and transitive relation, while a strict order is irreflexive and transitive. A terminating strict order \(>\) is said to be well-founded. The recursive path order on term graphs will be parameterized by a preorder of tops, a so-called precedence.

**Definition 6.10 (Precedence)**

The set of all tops with function symbols from \(\Sigma\) is denoted by \(\text{Top}_S\).\(^2\) A precedence is a preorder \(\sqsubseteq\) on \(\text{Top}_S\). The strict part and the equivalence part of \(\sqsubseteq\) are defined by \(\sqsubset = (\sqsubseteq - \sqsubseteq)\) and \(\equiv = (\sqsubseteq \cap \sqsubseteq)\).

For example, Figure 15 shows a precedence over the function symbols of Example 6.1. To define the recursive path order, we recall from [29] the lifting of an order to a multiset order. Let \(>\) be a strict order on a set \(A\). The multiset extension \(>^{\text{mul}}\) on the set of finite multisets over \(A\) is defined as follows: \(M >^{\text{mul}} N\) if there are multisets \(X\) and \(Y\) such that (1) \(\emptyset \neq X \subseteq M\), (2) \(N = (M - X) \cup Y\), and (3) for all \(y \in Y\) there is some \(x \in X\) with \(x > y\).

In the following definition, \(\mathcal{T}G_\Sigma\) denotes the set of all variable-free term graphs over \(\Sigma\).

**Definition 6.11 (Recursive path order)**

Let \(\sqsubseteq\) be a precedence. The recursive path order \(>_{\text{rpo}}\) on \(\mathcal{T}G_\Sigma\) is defined inductively as follows: \(G >_{\text{rpo}} H\) if

(1) \(S >_{\text{rpo}} H\) or \(S = H\) for some immediate subgraph \(S\) of \(G\), or

\(^2\)Note that by our convention to deal with standard term graphs only, the tops in \(\text{Top}_S\) are pairwise non-isomorphic.
(2) \( \text{top}_G \sqsupset \text{top}_H \) and \( G > \text{rpo} \ T \) for all immediate subgraphs \( T \) of \( H \), or

(3) \( \text{top}_G = \text{top}_H \) and \( \text{Sub}_G >^\text{rpo} \text{Sub}_H \).

A more general variant of the recursive path order can be found in [91], where the equality \( \text{top}_G = \text{top}_H \) in (3) is replaced by \( \text{top}_G \equiv \text{top}_H \). The equality \( S = H \) in (1) is relaxed similarly.

**Theorem 6.12**

The recursive path order is well-founded whenever its underlying precedence is well-founded.

To derive from this result a proof technique for the termination of \( \Rightarrow_\text{coll} \), we have to consider precedences containing the collapsing of tops.

**Definition 6.13**

A precedence \( \sqsupset \) is **collapse-compatible** if whenever there is a graph morphism \( t \to u \) for some \( t, u \in \text{Top}_G \), then \( t \sqsupset u \). If moreover \( \sqsupset \) is well-founded, then \( \sqsupset \) is a well-precedence.

The precedence of Figure 15, for example, is a well-precedence. Recall, for the following theorem, that a term graph \( L \) is an instance of a term \( t \) if there is a root preserving graph morphism \( \Diamond t \to L \). A variable-free instance is called a **ground instance**.

**Theorem 6.14**

Let \( >_\text{rpo} \) be induced by a well-precedence. Then \( \Rightarrow_\text{coll} \) is terminating if \( L \Rightarrow_\text{root} \Rightarrow_\text{rpo} \ R \) implies \( L >_\text{rpo} \ R \), for every rule \( l \to r \) in \( \mathcal{R} \) and every ground instance \( L \) of \( l \).

Using the precedence of Figure 15, we can convince ourselves that both rewrite rules of Example 6.1 satisfy the condition of Theorem 6.14. Thus, we obtain an alternative proof for the termination of \( \Rightarrow_\text{coll} \) over that system. We now give a further example for the use of the recursive path order.

**Example 6.3**

Consider the following rewrite system:

\[
\neg \text{true} \land \text{false} \to \neg \neg \text{true} \\
\neg \neg x \to x \land x \\
\neg \text{true} \to \text{false}
\]

Again, term rewriting is not terminating, which can be seen as follows:

\[
\neg \text{true} \land \text{false} \to \neg \neg \text{true} \to \neg \text{true} \land \neg \text{true} \to \neg \text{true} \land \text{false} \to \ldots
\]
However, termination of $\Rightarrow_{\text{coll}}$ can easily be checked by means of Theorem 6.14, using the well-precedence of Figure 16.

![Well-precedence](image)

Figure 16: A well-precedence for Example 6.3
7 Confluence

An important consequence of the completeness of $\Rightarrow_\text{coll}$ for equational proofs (Corollary 5.3) is that if $\Rightarrow_\text{coll}$ is convergent, validity of equations can be decided by a $\Rightarrow_\text{coll}$-based reduction procedure. In this section we take a look at the relation between $\Rightarrow_\text{coll}$ and $\Rightarrow$ with respect to confluence, which is just opposite to the relation with respect to termination: confluence of $\Rightarrow_\text{coll}$ strictly implies confluence of $\Rightarrow$. With respect to convergence, however, there is the same relation as in the case of termination. That is, $\Rightarrow_\text{coll}$ is convergent for more systems than $\Rightarrow$. Besides these issues, we address decidability and modularity of confluence, confluence of plain term graph rewriting, and confluence modulo bisimilarity.

7.1 The relation to term rewriting

We start with two counterexamples from [88] to demonstrate that confluence of term rewriting implies neither confluence of $\Rightarrow$ nor confluence of $\Rightarrow_\text{coll}$.

Example 7.1
Suppose that $\mathcal{R}$ is given as follows:\(^3\)

\[
\begin{align*}
\text{f(x)} & \Rightarrow \text{g(x,x)} \\
\text{a} & \Rightarrow \text{b} \\
\text{g(a,b)} & \Rightarrow \text{c} \\
\text{g(b,b)} & \Rightarrow \text{f(a)}
\end{align*}
\]

Using structural induction on terms, it can be shown that every term has a unique normal form. So term rewriting is normalizing and confluent. But Figure 17 shows that $\Rightarrow$ is neither normalizing nor confluent. The same applies to $\Rightarrow_\text{coll}$, as the collapse step $\Delta \text{g(b,b)} \Rightarrow \lor \text{g(b,b)}$ does not essentially change the situation. The problem here is that the sharing created by the rule $\text{f(x)} \Rightarrow \text{g(x,x)}$ prevents the rewrite step $\text{g(a,a)} \Rightarrow \text{g(a,b)}$, which is necessary to reduce $\text{g(a,a)}$ to $\text{c}$.

Example 7.2
In the case of $\Rightarrow_\text{coll}$, confluence is not even guaranteed over an orthogonal one-rule system. Consider the rule

\[
\text{a} \Rightarrow \text{f(a)}
\]

\(^3\)It is interesting to note that the same system was independently invented as a counterexample to completeness of (term-based) basic narrowing [77].
Figure 17: Non-confluence of $\Rightarrow$ and $\Rightarrow_{\text{coll}}$

and suppose that $\Sigma$ contains a binary function symbol $g$. Figure 18 shows two $\Rightarrow_{\text{coll}}$-derivations starting from $\Delta g(a, a)$, where the resulting graphs do not have a common reduct. (The graphs derivable on the left represent the terms $g(f^n(a), f^n(a))$, $n \geq 1$, while the graphs derivable on the right represent $g(f^n(a), f^{n+1}(a))$, $n \geq 0$.)

Figure 18: Non-confluence of $\Rightarrow_{\text{coll}}$

As mentioned above, confluence behaves opposite to termination in that it carries over from $\Rightarrow_{\text{coll}}$ to $\rightarrow$. This fact has a straightforward proof by the completeness of $\Rightarrow_{\text{coll}}$ for term convertibility.

**Theorem 7.1 ([88])**

If $\Rightarrow_{\text{coll}}$ is confluent, then $\rightarrow$ is confluent as well.

**Proof**

Let $\Rightarrow_{\text{coll}}$ be confluent and consider terms $s$, $t$ and $u$ such that $s \leftarrow^* t \rightarrow^* u$. Choose any term graphs $S$ and $U$ such that $\text{term}(S) = s$ and $\text{term}(U) = u$. Since $s \leftarrow^* u$, Corollary 5.3 gives $S \Rightarrow^*_{\text{coll}} U$. Hence, by confluence, there is some term graph $W$ such that $S \Rightarrow^*_{\text{coll}} W \Leftarrow^*_{\text{coll}} U$ (note that confluence is equivalent to the Church-Rosser property, see Lemma 2.4). By soundness of $\Rightarrow_{\text{coll}}$, this implies $s \rightarrow^* \text{term}(W) \Leftarrow^* u$. Thus $\rightarrow$ is confluent.

This result holds analogously for local confluence [89]. Note also that in Example 7.1, $\Rightarrow$ and $\Rightarrow_{\text{coll}}$ are not even locally confluent.

39
Despite the fact that confluence does not carry over from term to term graph rewriting, normal forms with respect to \( \Rightarrow_{\text{coll}} \) are unique if and only if term normal forms are unique. In other words, the possible non-confluence of \( \Rightarrow_{\text{coll}} \) over a confluent term rewriting system has no impact on the uniqueness of normal forms. To prove this, we need the following lemma.

**Lemma 7.2**
A term graph \( G \) is a normal form with respect to \( \Rightarrow_{\text{coll}} \) if and only if \( G \) is fully collapsed and \( \text{term}(G) \) is a normal form with respect to \( \rightarrow \).

Recall that an abstract reduction system \( (A, \rightarrow) \) has *unique normal forms* if whenever \( a \leftrightarrow^* b \) for normal forms \( a \) and \( b \), then \( a = b \).

**Theorem 7.3 ([89])**
The relation \( \Rightarrow_{\text{coll}} \) has unique normal forms if and only if \( \rightarrow \) has unique normal forms.

**Proof**
Let \( \Rightarrow_{\text{coll}} \) have unique normal forms. Consider term normal forms \( t \) and \( u \) such that \( t \leftrightarrow^* u \). Then \( \forall t \leftrightarrow^* \text{term}(G) \) by completeness of \( \Rightarrow_{\text{coll}} \). By Lemma 7.2, \( \forall t \) and \( \forall u \) are normal forms with respect to \( \Rightarrow_{\text{coll}} \). Hence, by uniqueness of normal forms, \( \forall t = \forall u \). With Corollary 3.12 follows \( t = u \).

Conversely, suppose that \( \rightarrow \) has unique normal forms. Let \( G \) and \( H \) be term graph normal forms such that \( G \leftrightarrow^* \text{term}(G) \). Then \( \text{term}(G) \leftrightarrow^* \text{term}(H) \) by soundness of \( \Rightarrow_{\text{coll}} \), and both terms are normal forms by Lemma 7.2. Thus, uniqueness of normal forms gives \( \text{term}(G) = \text{term}(H) \). Hence \( G = \forall \text{term}(G) = \forall \text{term}(H) = H \) by the uniqueness of fully collapsed term graphs (Lemma 3.9).

As a consequence of this result, confluence carries over from term rewriting to \( \Rightarrow_{\text{coll}} \) if the latter is normalizing. In particular, \( \Rightarrow_{\text{coll}} \) is convergent whenever term rewriting is convergent.

**Corollary 7.4**

1. Suppose that \( \Rightarrow_{\text{coll}} \) is normalizing. Then \( \Rightarrow_{\text{coll}} \) is confluent if and only if \( \rightarrow \) is confluent.

2. If \( \rightarrow \) is convergent, then \( \Rightarrow_{\text{coll}} \) is convergent as well.

**Proof**

1. The "only if"-direction is contained in Theorem 7.1. Conversely, let \( \Rightarrow_{\text{coll}} \) be normalizing and \( \rightarrow \) be confluent. The latter implies that \( \rightarrow \) has unique normal forms (see Lemma 2.4). Hence, by Theorem 7.3, \( \Rightarrow_{\text{coll}} \) has unique normal forms as well. But it is easy to verify that a normalizing relation with unique normal forms is confluent.
(2) By Theorem 6.1, termination of $\rightarrow$ implies termination of $\Rightarrow_{\text{coll}}$. In particular, $\Rightarrow_{\text{coll}}$ is normalizing then. Hence, by (1), confluence carries over from $\rightarrow$ to $\Rightarrow_{\text{coll}}$.

The next example shows that the converse of the second part of Corollary 7.4 does not hold. That is, the class of term rewriting systems over which $\Rightarrow_{\text{coll}}$ is convergent strictly contains the class of convergent term rewriting systems.

**Example 7.3**
Consider the following system:

\[
\begin{align*}
  f(x) & \rightarrow g(x, x) \\
  a & \rightarrow b \\
  g(a, b) & \rightarrow f(a)
\end{align*}
\]

Again it can be shown that every term has a unique normal form, implying that term rewriting is confluent. But $\rightarrow$ is not terminating, as there is the following infinite rewrite sequence:

\[f(a) \rightarrow g(a, a) \rightarrow g(a, b) \rightarrow f(a) \rightarrow \ldots\]

Using the recursive path order induced by the well-precedence of Figure 19, it is not difficult to see that Theorem 6.14 ensures termination of $\Rightarrow_{\text{coll}}$. Then Corollary 7.4(1) guarantees that $\Rightarrow_{\text{coll}}$ is confluent, and hence convergent.

![Figure 19: A well-precedence for Example 7.3](image)

### 7.2 Decidability and combined systems

Analogously to the situation for term rewriting [68], termination of $\Rightarrow_{\text{coll}}$ implies that confluence can be decided by an analysis of so-called critical pairs. We state the decidability of confluence without entering into the technicalities of critical pairs for term graph rewriting, which can be found in [89, 90].

**Theorem 7.5 ([89])**

*There is an algorithm that solves the following problem:*
Instance: A finite term rewriting system $\mathcal{R}$ such that $\Rightarrow_{\text{coll}}$ is terminating.

Question: Is $\Rightarrow_{\text{coll}}$ confluent?

Note that by the combination of this result with Corollary 7.4(1), if $\Rightarrow_{\text{coll}}$ is terminating, then confluence of term rewriting is decidable as well.

Next we consider confluence of $\Rightarrow_{\text{coll}}$ over combined systems. In contrast to confluence of term rewriting [99], confluence of $\Rightarrow_{\text{coll}}$ is not a modular property. In fact, confluence may be destroyed just by extending the set $\Sigma$ of function symbols. This can be seen from Example 7.2, where $\Rightarrow_{\text{coll}}$ is confluent when $\Sigma = \{a, f\}$. After adding a binary symbol $g$, confluence breaks down.

We will see, however, that convergence of $\Rightarrow_{\text{coll}}$ is preserved by the union of two crosswise disjoint systems if their left-hand sides do not mutually overlap.

Definition 7.6 (Non-interfering systems)

Two term rewriting system $\mathcal{R}_0$ and $\mathcal{R}_1$ are non-interfering if no left-hand side of $\mathcal{R}_i$ overlaps a left-hand side of $\mathcal{R}_{1-i}$, for $i = 0, 1$.

As in Subsection 6.2, we write $\Rightarrow_\mathcal{R}$ for the relation $\Rightarrow_{\text{coll}}$ over a term rewriting system $\mathcal{R}$.

Theorem 7.7 ([88])

Let $\mathcal{R}_0 \cup \mathcal{R}_1$ be the union of two crosswise disjoint and non-interfering term rewriting systems. If $\Rightarrow_\mathcal{R}_0$ and $\Rightarrow_\mathcal{R}_1$ are convergent, then $\Rightarrow_\mathcal{R}_0 \cup \mathcal{R}_1$ is convergent as well.

Recently, this result was extended by relaxing crosswise disjointness [72]. The preservation of convergence contrasts with the situation for term rewriting, where even disjoint unions need not preserve this property. The following counterexample was given in [31].

Example 7.4

Consider the following two systems with disjoint function symbols:

$$\mathcal{R}_0 = \begin{cases} f(0,1,x) & \to f(x,x,x) \\ f(x,y,z) & \to 2 \\ 0 & \to 2 \\ 1 & \to 2 \end{cases}$$

$$\mathcal{R}_1 = \begin{cases} g(x,y,y) & \to x \\ g(x,x,y) & \to y \end{cases}$$
It can be shown that term rewriting is convergent for both systems. But the union $\mathcal{R}_0 \cup \mathcal{R}_1$ is not terminating:

$$f(g(0,1,1),g(0,1,1),g(0,1,1)) \rightarrow f(0,g(0,1,1),g(0,1,1))$$
$$\rightarrow^2 f(0,g(2,1,1),g(0,1,1))$$
$$\rightarrow f(0,1,g(0,1,1))$$
$$\rightarrow f(g(0,1,1),g(0,1,1),g(0,1,1))$$
$$\rightarrow \ldots$$

Theorem 7.7 shows that $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is convergent, since $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are convergent by Corollary 7.4(2). In particular, every exhaustive rewrite sequence starting from $\Delta f(g(0,1,1),g(0,1,1),g(0,1,1))$ ends in the unique normal form $\Delta^2$.

### 7.3 Plain term graph rewriting and confluence modulo bisimilarity

So far we have concentrated on confluence of term graph rewriting with collapsing. This subsection presents some confluence results for plain term graph rewriting and for the relations $\Rightarrow_{bi}$ and $\Rightarrow_{\infty}$. (For the first two theorems, the reader may wish to look up Definition 5.5 which introduces orthogonal systems.)

**Theorem 7.8 ([95])**

*If $\mathcal{R}$ is orthogonal, then $\Rightarrow$ is subcommutative.*

This result was proved in a technical framework which slightly differs from the present one. The proof actually shows that $\Rightarrow$ is subcommutative for the larger class of non-overlapping term rewriting systems. Beyond non-overlapping systems, however, there is virtually no significant class of systems for which $\Rightarrow$ is confluent. By the next example, $\Rightarrow$ need not be confluent even for left-linear, convergent term rewriting systems.

**Example 7.5**

The left-linear system

$$f(x) \rightarrow g(x,x)$$
$$f(a) \rightarrow g(a,a)$$

is clearly convergent under term rewriting, but Figure 20 shows that plain term graph rewriting is not confluent.

This example suggests to consider a version of confluence where joining derivations need not end in the same graph but only in bisimilar graphs.
Definition 7.9 (Confluence modulo bisimilarity)
A binary relation $\Rightarrow$ on term graphs is **confluent modulo bisimilarity** if whenever $G_1 \leq^* G \Rightarrow^* H \leq^* H_1$, there are term graphs $G_2$ and $H_2$ such that $G_1 \Rightarrow^* G_2 \sim H_2 \leq^* H_1$.

For plain term graph rewriting, confluence modulo bisimilarity and confluence are incomparable [7]. But both properties hold for orthogonal systems.

**Theorem 7.10 ([7])**
If $\mathcal{R}$ is orthogonal, then $\Rightarrow$ is confluent modulo bisimilarity.

It is worth mentioning that “orthogonal” cannot be generalized to “non-overlapping”. To ensure that $\Rightarrow$ is confluent modulo bisimilarity for a confluent, non-orthogonal term rewriting system, $\Rightarrow$ has to be normalizing.

**Theorem 7.11 ([7])**
If $\mathcal{R}$ is left-linear, $\rightarrow$ confluent and $\Rightarrow$ normalizing, then $\Rightarrow$ is confluent modulo bisimilarity.

Here normalization of $\Rightarrow$ cannot be relaxed to normalization of $\rightarrow$, as is witnessed by Example 7.1, and left-linearity cannot be dropped either. Moreover, the result can be strengthened in that $\Rightarrow$ is even *Church-Rosser modulo bisimilarity*, see [6].

The last result in this section shows that $\Rightarrow_{bi}$ and $\Rightarrow_{\sim}$ are confluent exactly for all confluent term rewriting systems.

**Theorem 7.12 ([7, 6])**
The following are equivalent:

1. $\Rightarrow_{bi}$ is confluent.
2. $\Rightarrow_{\sim}$ is confluent.
3. $\rightarrow$ is confluent.
We finally mention that for $\Rightarrow_{\text{coll}}$, $\Rightarrow_{\text{copy}}$ and $\Rightarrow_{\text{bi}}$, confluence modulo bisimilarity is equivalent to confluence. More generally, this applies to every relation on term graphs the transitive closure of which contains $\succ$ or $\prec$ (see [6]).
8 Term Graph Narrowing

Narrowing combines term rewriting with unification in order to solve equations: given an equation \( s \approx t \), the goal is to generate a substitution \( \sigma \) such that \( \mathcal{R} \models \sigma(s) \approx \sigma(t) \). Narrowing originates from the area of theorem proving and is used as an operational principle for combining functional and logic programming. See [48] for a survey of the latter application.

In this section we study term graph narrowing, a graph-based form of narrowing which combines term graph rewriting with term unification. As in the case of rewriting, the motivation for sharing (and collapsing) common subexpressions is to improve the efficiency of computations in time and space.

Narrowing is said to be complete if for every solution of an equation, it can generate a solution that is at least as general. While conventional narrowing is complete whenever term rewriting is normalizing and confluent, for term graph narrowing one has to require that \( \Rightarrow_{\text{coll}} \) is normalizing and confluent. In Subsection 8.2 we discuss the completeness of two restricted forms of term graph narrowing, called minimally collapsing and maximally collapsing narrowing. Our presentation is based on [46, 45].

8.1 Term graph narrowing

We will need substitutions replacing variables in term graphs by term graphs. A pair \( x/G \) consisting of a variable \( x \) and a term graph \( G \) is a substitution pair. It is applied to an \( x \)-labelled edge \( e \) in a term graph \( H \) by removing \( e \), adding (disjointly) \( G \), and identifying \( \text{res}(e) \) with \( \text{root}_G \).

Definition 8.1 (Term graph substitution)

A term graph substitution is a finite set \( \alpha = \{x_1/G_1, \ldots, x_n/G_n\} \) of substitution pairs such that \( x_1, \ldots, x_n \) are pairwise distinct and \( x_i \not= \text{term}(G_i) \) for \( i = 1, \ldots, n \). Given a term graph \( H \), applying \( x_1/G_1, \ldots, x_n/G_n \) simultaneously to all edges labelled with \( x_1, \ldots, x_n \) yields the term graph \( H_\alpha \).

The domain of \( \alpha \) is the set \( \text{Dom}(\alpha) = \{x_1, \ldots, x_n\} \), and its composition with a term graph substitutions \( \beta \) is defined by

\[
\alpha \beta = \{x/G\beta \mid x/G \in \alpha \text{ and } x \not= \text{term}(G\beta)\} \cup \{y/H \in \beta \mid y \not\in \text{Dom}(\alpha)\}
\]

to satisfy \( H(\alpha \beta) = (H\alpha)\beta \) for every term graph \( H \).

A term graph substitution \( \alpha \) induces the term substitution \( \alpha^{\text{term}} : \mathbb{T}_{\Sigma,X} \rightarrow \mathbb{T}_{\Sigma,X} \) mapping \( x_i \) to \( \text{term}(G_i) \), for \( i = 1, \ldots, n \), and each other variable to itself. Given
a term substitution \( \sigma \) and a term \( t \), we will write \( t\sigma \) in place of \( \sigma(t) \). We may represent \( \sigma \) by the set \( \{ x_i/t_i, \ldots, x_n/t_n \} \) if \( x_i\sigma = t_i \) for \( i = 1, \ldots, n \) and \( x\sigma = x \) for each other variable \( x \).

A variant of a term rewrite rule \( l \rightarrow r \) is a rule of the form \( l\sigma \rightarrow r\sigma \), where \( \sigma \) is an injective substitution mapping variables to variables. A set of terms \( \{ t_1, \ldots, t_n \} \) is unifiable if there is a substitution \( \sigma \) such that \( t_1\sigma = t_2\sigma = \cdots = t_n\sigma \). In this case \( \sigma \) can be chosen as a most general unifier, meaning that for every substitution \( \tau \) with \( t_1\tau = t_2\tau = \cdots = t_n\tau \) there exists a substitution \( \rho \) such that \( \tau = \rho \circ \sigma \) (see for example [11]).

**Definition 8.2 (Term graph narrowing)**

Let \( G \) and \( H \) be term graphs, \( U \) a set of non-variable nodes in \( G \), \( l \rightarrow r \) a variant of a rule in \( \mathcal{R} \), and \( \alpha \) a term graph substitution. There is a narrowing step \( G \leadsto_{U, l \rightarrow r, \alpha} H \) if a term is a most general unifier of \( \{ \text{term}_G(u) \mid u \in U \} \cup \{ l \} \), and

\[
G\alpha \geq G' \implies H
\]

for some collapsing \( c: G\alpha \rightarrow G' \) such that \( U = \{ \overline{v} \mid c(\overline{v}) = v \} \).

We denote such a step also by \( G \leadsto_{\alpha} H \). A term graph narrowing derivation is sequence of the form \( G = G_1 \leadsto_{\alpha_1} G_2 \leadsto_{\alpha_2} \cdots \leadsto_{\alpha_{n-1}} G_n = H \). It may be denoted by \( G \leadsto_{\alpha}^* H \), where \( \alpha = \alpha_1\alpha_2\cdots\alpha_{n-1} \) if \( n \geq 2 \) and \( \alpha = \emptyset \) if \( n = 1 \).

From now on we assume that \( \mathcal{R} \) contains the rule \( x =? x \rightarrow \text{true} \), where the binary symbol \( =? \) and the constant \text{true} do not occur in any other rule. A goal is a term of the form \( s =? t \) such that \( s \) and \( t \) do not contain \( =? \) and \text{true}. A solution of this goal is a substitution \( \sigma \) satisfying \( s\sigma \leftrightarrow t\sigma \) (equivalently: \( \mathcal{R} \models s\sigma \approx t\sigma \)).

**Example 8.1**

Let \( \mathcal{R} \) consist of the following rules:

\[
\begin{align*}
0 + x & \rightarrow x \\
\text{s}(x) + y & \rightarrow \text{s}(x + y) \\
0 \times x & \rightarrow 0 \\
\text{s}(x) \times y & \rightarrow (x \times y) + y \\
x =? x & \rightarrow \text{true}
\end{align*}
\]

Suppose that we want to solve the goal \((z \times z) + (z \times z) =? \text{s}(z)\). Figure 21 shows a term graph narrowing derivation starting from the fully collapsed term graph representing this goal. The table below the derivation gives the applied rewrite rules and the involved term substitutions. In each step, the set \( U \) of Definition 8.2 is a singleton. Note that steps (c), (d) and (e) are nothing but \( \Rightarrow \)-steps, and that step (f) consists of a collapse step followed by a \( \Rightarrow \)-step. The derivation
computes the term substitution \( \{x/0, x'/s(0), y/s(0), z/s(0)\} \). Restricting this substitution to the variables of the goal yields the solution \( \{z/s(0)\} \). Solving the same goal by term-based narrowing requires nine steps, demonstrating that term graph narrowing can speed up the computation of solutions.

Given substitutions \( \sigma \) and \( \tau \), and \( V \subseteq X \), we write \( \sigma =_R \tau [V] \) if \( x\sigma \leftrightarrow^* x\tau \) for each \( x \in V \), and \( \sigma \leq_R \tau [V] \) if there is a substitution \( \rho \) such that \( \sigma\rho =_R \tau [V] \). The set of variables occurring in a term graph \( G \) is denoted by \( \text{Var}(G) \), that is, \( \text{Var}(G) = \text{lab}_G(E_G) \cap X \).

**Theorem 8.3 (Soundness and completeness of narrowing)**

Let \( G \) be a term graph such that \( \text{term}(G) \) is a goal \( s =^? t \).

1. If \( G \twoheadrightarrow^* \Delta \text{true} \), then \( \alpha \text{term} \) is a solution of \( s =^? t \).

2. If \( \Rightarrow_{\text{coll}} \) is normalizing and confluent, then for every solution \( \sigma \) of \( s =^? t \) there exists a narrowing derivation \( G \twoheadrightarrow^* \Delta \text{true} \) such that \( \beta \text{term} \leq_R \sigma [\text{Var}(G)] \).

In the sequel, we will refer to the conclusion of statement (2) as *completeness* of term graph narrowing.

**Example 8.2**

This example shows that term graph narrowing is not complete in general for a confluent and normalizing term rewriting system, although term-based narrowing is complete for such systems [78]. As a counterexample we can use the system of Example 7.1:

\[
\begin{align*}
  f(x) & \rightarrow g(x,x) \\
  a & \rightarrow b \\
  g(a,b) & \rightarrow c \\
  g(b,b) & \rightarrow f(a)
\end{align*}
\]

After adding the rule \( x =^? x \rightarrow \text{true} \), term rewriting remains normalizing and confluent. Since \( f(a) \) is reducible to \( c \), the empty substitution is a solution of the goal \( f(a) =^? c \). But Figure 22 demonstrates that term graph narrowing—which for a variable-free goal is just a combination of collapsing and rewriting—cannot solve the goal \( f(a) =^? c \). Note that in this example, \( \Rightarrow_{\text{coll}} \) is neither normalizing nor confluent (see Figure 17 for the absence of confluence).
Figure 21: A term graph narrowing derivation
8.2 Minimally and maximally collapsing narrowing

In this subsection we consider the completeness of two restricted forms of term graph narrowing where all steps contain a minimal or maximal collapsing, respectively.

**Definition 8.4 (Minimal collapsing)**
A collapsing $G \triangleright M$ is minimal with respect to a redex $(u, l \rightarrow r)$ in $M$ if for each term graph $M'$ with $G \triangleright M' \triangleright M$ and each preimage $v'$ of $v$ in $M'$, the pair $(v', l \rightarrow r)$ is not a redex.

In particular, if $G$ equals $M$, then $G \triangleright M$ is minimal since no $M'$ with $G \triangleright M' \triangleright M$ exists. A proper collapsing $G \triangleright M$ is minimal only if $l \rightarrow r$ is not left-linear and cannot be applied at any preimage of $v$ in $G$.

**Definition 8.5 (Minimally collapsing narrowing)**
A term graph narrowing derivation is minimally collapsing if for each narrowing step $G \rightarrow Ga \triangleright G' \Rightarrow (v, l \rightarrow r) H$, the collapsing $Ga \triangleright G'$ is minimal with respect to the redex $(v, l \rightarrow r)$.

For example, the derivation of Figure 21 is minimally collapsing. Note that in a minimally collapsing step $G \rightarrow_{U,l \rightarrow r,\alpha} H$, the set $U$ must be a singleton. It turns out that Theorem 8.3 can be strengthened by replacing unrestricted term graph narrowing with minimally collapsing narrowing.

**Theorem 8.6 (Completeness of minimally collapsing narrowing)**
Minimally collapsing narrowing is complete whenever $\Rightarrow_{\text{coll}}$ is normalizing and confluent.
We now turn to maximally collapsing narrowing, that is, we consider narrowing derivations in which all involved collapse steps yield fully collapsed term graphs.

**Definition 8.7 (Maximally collapsing narrowing)**
A term graph narrowing derivation is *maximally collapsing* if for each narrowing step \( G \Rightarrow G' \) with \( G \subseteq G' \Rightarrow H \), the term graph \( G' \) is fully collapsed.

**Example 8.3**
Consider the rules

\[
\begin{align*}
\text{exp}(0) & \rightarrow s(0) \\
\text{exp}(s(x)) & \rightarrow \text{exp}(x) + \text{exp}(x)
\end{align*}
\]

specifying the function \( \text{exp}: n \mapsto 2^n \) on natural numbers. Figure 23 demonstrates that maximally collapsing narrowing can solve a goal of the form

\[
\text{exp}(x) = \underbrace{s(0) + \cdots + s(0)}_{2^n \text{-times}}
\]

in \( n + 2 \) steps if the goal is suitably represented. (Substitutions are represented only by those parts affecting the variables in the graphs.) In contrast, both tree-based narrowing and minimally collapsing narrowing need a number of steps exponential in \( n \) to solve such a goal.

While minimally collapsing narrowing is complete when term graph rewriting is normalizing and confluent, the following counterexample shows that this is not the case for maximally collapsing narrowing.

**Example 8.4**
Consider the following term rewriting system:

\[
\begin{align*}
g(x, y) & \rightarrow g(a, a) \\
a & \rightarrow b \\
g(a, b) & \rightarrow b \\
x = x & \rightarrow \text{true}
\end{align*}
\]

Here \( \Rightarrow_{\text{coll}} \) is normalizing and confluent, which can be shown by induction on the size of term graphs. But the tree \( \Delta g(a, a) \) cannot be reduced to its normal form \( \Delta b \) if proper rewrite steps are preceded by maximal collapse steps, as shown in Figure 24. It follows that maximally collapsing narrowing cannot solve the goal \( g(a, a) = b \), although \( g(a, a) \) and \( b \) are clearly equivalent.

Completeness of maximally collapsing narrowing can be ensured, however, by strengthening normalization to termination.

51
Figure 23: A maximally collapsing narrowing derivation
Theorem 8.8 (Completeness of maximally collapsing narrowing)

Maximally collapsing narrowing is complete whenever $\Rightarrow_{\text{coll}}$ is convergent.

Note that as a corollary of this result, maximally collapsing narrowing is complete for all convergent term rewriting systems.

8.3 Bibliographic notes

The present definition of term graph narrowing was introduced in [46]. It extends the definition in [45] in that the latter corresponds to the special case where all nodes in the set $U$ represent the same term. The paper [46] also studies basic term graph narrowing, an analogue to basic term-based narrowing [55]. Roughly speaking, this strategy forbids narrowing steps at nodes that have been created by the substitutions of previous steps. It turns out that minimally collapsing basic narrowing is complete if $\Rightarrow_{\text{coll}}$ is innermost normalizing and confluent, and that Theorem 8.8 holds for maximally collapsing basic narrowing as well.

Basic narrowing on term graphs is also addressed in [70], but the kind of narrowing used there (going back to [44]) does not provide for a collapsing between the application of the unifier and the rewrite step. As a consequence, narrowing is incomplete for a non-left-linear system like \{f(x, x) → a\} (consider, for example, the goal $f(x, y) =? a$).

Narrowing on jungles, using conditional rewrite rules, is considered in [25]. Narrowing steps are based on jungle pushouts, leading to a kind of minimally collapsing narrowing. The results in [25] aim at showing the correctness of a concrete implementation of conditional narrowing. In [32], narrowing on possibly cyclic term graphs is studied, and an optimal strategy is given for the class of constructor-based orthogonal term rewriting systems.
9 Further Topics

We briefly mention some topics in term graph rewriting that have not been discussed in the preceding sections.

Optimality of reduction strategies—in the sense of finding a normal form in a minimal number of steps—is investigated in [95, 96, 97, 76]. Essential for these considerations is the subcommutativity of plain term graph rewriting over orthogonal systems.

There are a few papers describing implementations of term graph rewriting. So-called concurrent term rewriting is addressed in [40, 65], while [60, 39, 13] deal with the term graph rewrite language Dactl. A report on an implementation which enforces full collapsing can be found in [58].

Aspects of term graph rewriting relevant for the design, implementation and analysis of functional programming languages are discussed in [85, 3, 2]. In [49] the technique of memoization, which keeps computed values for later use, is realized by term graph rewriting.

In [22, 24, 23] it is shown how to simulate logic programming by term graph rewriting.

While this survey is restricted to acyclic term graphs and finitary term rewriting, one may also consider cyclic graphs and infinitary term rewriting. The interested reader may consult [36, 37, 63, 64, 18, 20].

Further issues that have been considered are term graph rewriting over conditional term rewriting systems [82], the relation between term graph rewriting and event structures [62, 17], and the term-generating power of context-free “jungle grammars” [35].

The area of graph reduction for the lambda calculus is related to term graph rewriting, but is beyond the scope of this survey. For information about this topic, we refer to [8, 5] and the references given there.
References


