# Chapter 1

# TERM GRAPH REWRITING

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

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 chapter surveys (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.

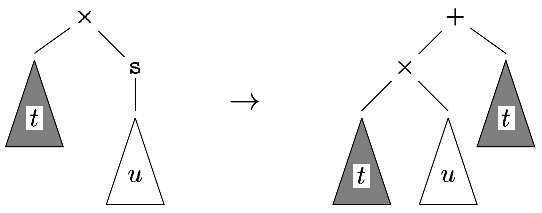
## Contents

## 1.1 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 $\mathtt{s}$ denotes the successor function on natural numbers):

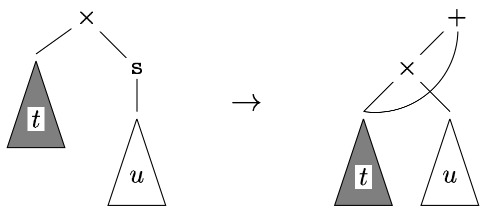
$$\begin{aligned} \mathtt{x} \times 0 &\to 0 \\ \mathtt{x} \times \mathtt{s}(\mathtt{y}) &\to (\mathtt{x} \times \mathtt{y}) + \mathtt{x} \end{aligned}$$

In applying the second rule to an expression of the form $t \times \mathtt{s}(u)$, 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 chapter 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 1.9. 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 chapter. The chapter was partly written during the author's stay at the University of Nijmegen in 1998. Support of this visit by the Deutsche Forschungsgemeinschaft is gratefully acknowledged.

## 1.2 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 chapter 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 [81] and systematically applied in influential papers of Rosen [95] 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 [9,11,16,28,52,56,67]. As the terminology in the literature is not completely uniform, alternative terms are given in parentheses.

**Definition 1.2.1 (Abstract reduction system)**
An *abstract reduction system* $\langle A, \to \rangle$ consists of a set $A$ and a binary relation $\to$ on $A$.

For the rest of this section, let $\langle A, \to \rangle$ be an arbitrary abstract reduction system. Given two elements $a$ and $b$ in $A$ such that $\langle a, b \rangle \in \to$, this is denoted by $a \to b$. The inverse relation of $\to$ is denoted by $\leftarrow$, and the compositon of two binary relations $\to_1$ and $\to_2$ on $A$ is written $\to_1 \circ \to_2$.

**Definition 1.2.2**

(1) The *identity* on $A$ is the relation $\to^0 \,=\, \{\langle a, a \rangle \mid a \in A\}$.

(2) The *reflexive closure* of $\to$ is the relation $\to^= \,=\, \to \cup \to^0$.

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

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

(5) The *transitive-reflexive closure* of $\to$ is the relation $\to^* \,=\, \to^+ \cup \to^0$.

(6) The *symmetric closure* of $\to$ is the relation $\leftrightarrow \,=\, \to \cup \leftarrow$.

(7) The *equivalence closure* of $\to$ (or *convertibility* with respect to $\to$), denoted by $\leftrightarrow^*$, 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 $a \to^* c \leftarrow^* b$ for some $c$. An element $a$ is a *normal form* if there is no $b$ such that $a \to b$, and it *has* a normal form if $a \to^* b$ for some normal form $b$. In the latter case $b$ is a normal form *of* $a$.

**Definition 1.2.3 (Termination and confluence properties)**
The relation $\to$ is

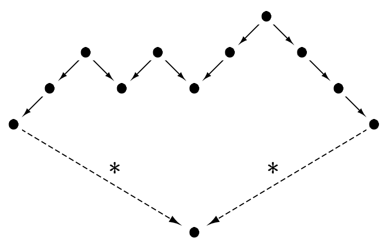
(1) *terminating* (or *strongly normalizing* or *noetherian*) if there does not exist an infinite sequence of the form $a_1 \to a_2 \to a_3 \to \dots$,
(2) *normalizing* (or *weakly normalizing*) if each element in $A$ has a normal form,
(3) *Church-Rosser* if for all $a$ and $b$ with $a \leftrightarrow^* b$ there is some $c$ such that $a \to^* c \leftarrow^* b$ (see Figure 1.1(a)),
(4) *confluent* if for all $a$, $b$ and $c$ with $b \leftarrow^* a \to^* c$ there is some $d$ such that $b \to^* d \leftarrow^* c$ (see Figure 1.1(b)),
(5) *locally confluent* (or *weakly confluent*) if for all $a$, $b$ and $c$ with $b \leftarrow a \to c$ there is some $d$ such that $b \to^* d \leftarrow^* c$ (see Figure 1.1(c)),
(6) *subcommutative* if for all $a$, $b$ and $c$ with $b \leftarrow a \to c$ there is some $d$ such that $b \to^= d \leftarrow^= c$ (see Figure 1.1(d)),
(7) *convergent* if it is terminating and confluent.

The following lemma lists some relations between the properties introduced above.

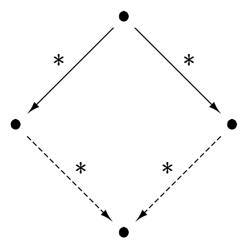
**Lemma 1.2.4**
(1) *Termination implies normalization.*
(2) *The Church-Rosser property is equivalent to confluence.*
(3) *Subcommutativity implies confluence.*
(4) *Confluence implies local confluence.*
(5) *Confluence implies uniqueness of normal forms, that is, whenever $a \leftrightarrow^* b$ for 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 $\to$ is subcommutative, then for all $a$, $b$ and $c$ with $b \leftarrow a \to^* c$ there is some $d$ such that $b \to^* d \leftarrow^* c$, while the second induction shows that the latter property implies confluence. Finally, it is easy to see that (5) follows from (2). $\square$

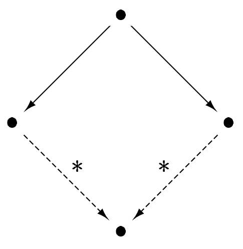
Notice that by statement (5), in a confluent relation every element has at most one normal form. The converses of the implications (1), (3), (4) and (5) do not



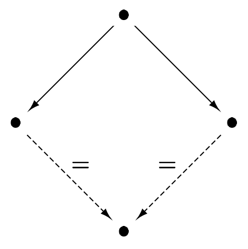
(a) Church-Rosser property



(b) confluence



(c) local confluence



(d) subcommutativity

Figure 1.1: Confluence properties

hold. Figure 1.2 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 ashort proof of this fact.)

**Lemma 1.2.5 (Newman's Lemma [81])**
*A terminating relation is confluent if and only if it is locally confluent.* □

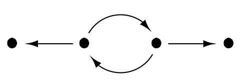


Figure 1.2: Local confluence without confluence

## 1.3 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.

### *1.3.1 From Hypergraphs to Term Graphs*

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

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

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

A *path* is an alternating sequence $\langle v_0, i_1, v_1, i_2, \dots, i_n, v_n \rangle$ $(n \geq 0)$ of nodes and positive integers such that for $j = 1, \dots, n$, there is an edge $e$ such that $\mathrm{res}(e) = v_{j-1}$ and $v_j$ is the $i_j^{\mathrm{th}}$ node in $\mathrm{arg}(e)$. 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 1.3.1 (Term graph)**
A graph $G$ is a *term graph* if

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

(2) $G$ is acyclic, and

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

Figure 1.3 shows a term graph with binary function symbols $+$ and $\times$, a unary function symbol $\mathtt{s}$, a constant $\mathtt{0}$ and a variable $\mathtt{y}$. 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 1.3, 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.

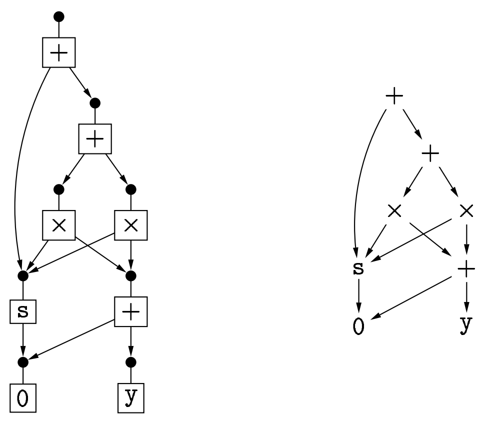


Figure 1.3: A term graph, depicted in two different ways

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 1.3.2 (Term representation)**

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

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

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

Note that the recursion in the above definition ends because term graphs are acyclic. For example, if $G$ is the term graph of Figure 1.3, then

$$\mathrm{term}(G) = +(\mathtt{s}(\mathtt{0}), +(\times(\mathtt{s}(\mathtt{0}), +(\mathtt{0}, \mathtt{y})), \times(\mathtt{s}(\mathtt{0}), +(\mathtt{0}, \mathtt{y})))).$$

Using infix notation for $+$ and $\times$, this term reads

$$\mathtt{s(0)} + ((\mathtt{s(0)} \times (\mathtt{0} + \mathtt{y})) + (\mathtt{s(0)} \times (\mathtt{0} + \mathtt{y}))).$$

A *graph morphism* $f \colon G \to H$ between two graphs $G$ and $H$ consists of two functions $f_{\mathrm{V}} \colon \mathrm{V}_G \to \mathrm{V}_H$ and $f_{\mathrm{E}} \colon \mathrm{E}_G \to \mathrm{E}_H$ that preserve labels and attachment to nodes, that is, $\mathrm{lab}_H \circ f_{\mathrm{E}} = \mathrm{lab}_G$ and $\mathrm{att}_H \circ f_{\mathrm{E}} = f_{\mathrm{V}}^* \circ \mathrm{att}_G$ (where $\circ$ denotes function composition and $f_{\mathrm{V}}^* \colon \mathrm{V}_G^* \to \mathrm{V}_H^*$ maps a string $v_0 \ldots v_n$ to $f_{\mathrm{V}}(v_0) \ldots f_{\mathrm{V}}(v_n)$). The morphism $f$ is *injective* (*surjective*) if $f_{\mathrm{V}}$ and $f_{\mathrm{E}}$ 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 $\langle i_1, i_2, \ldots, i_n \rangle$ such that there exists a path $\langle v_0, i_1, v_1, i_2, \ldots, i_n, v_n \rangle$ with $v_0 = \mathrm{root}_G$ and $v_n = v$. We denote by $\mathrm{Acc}(v)$ the set of all access paths of $v$.

**Definition 1.3.3 (Standard term graph)**
A term graph $G$ is a *standard term graph* if

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

An example for the naming of nodes in a standard term graph is given in Figure 1.4.

For every term graph we can construct an isomorphic standard term graph by replacing each node $v$ with $\mathrm{Acc}(v)$ and modifying the edge set and the labelling and attachment functions correspondingly. Moreover, given a graph morphism $f \colon G \to H$ between term graphs such that $f_{\mathrm{V}}(\mathrm{root}_G) = \mathrm{root}_H$, we have $\mathrm{Acc}(v) \subseteq \mathrm{Acc}(f_{\mathrm{V}}(v))$ 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.

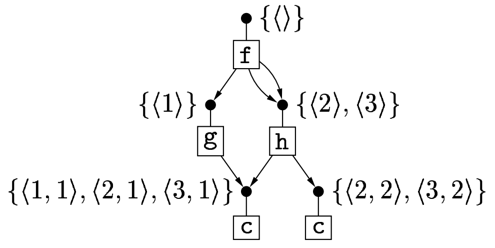


Figure 1.4: A standard term graph

**Lemma 1.3.4**
*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.

### *1.3.2 Collapsing, Copying and Bisimilarity*

By the three conditions of Definition 1.3.1, a graph morphism $f\colon G \to H$ between term graphs $G$ and $H$ is surjective if and only if $f_V(\mathrm{root}_G) = \mathrm{root}_H$. In this case $H$ can be seen as a "collapsed" or "compressed" version of $G$.

**Definition 1.3.5 (Collapsing and copying)**
Given two term graphs $G$ and $H$, $G$ *collapses* to $H$ if there is a graph morphism $G \to H$ mapping $\mathrm{root}_G$ to $\mathrm{root}_H$. 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 1.5.

**Lemma 1.3.6**
*For all term graphs $G$ and $H$, $G \succeq H$ implies* $\mathrm{term}(G) = \mathrm{term}(H)$.

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

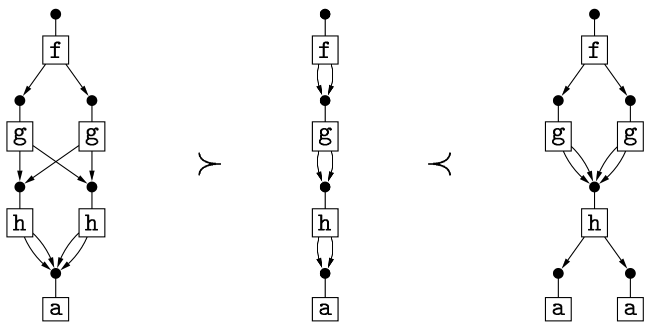


Figure 1.5: Collapsing and copying

$$\begin{aligned}
\mathrm{term}_G(v) &= \mathrm{lab}_G(e)(\mathrm{term}_G(v_1), \ldots, \mathrm{term}_G(v_n)) \\
&= \mathrm{lab}_H(f_{\mathrm{E}}(e))(\mathrm{term}_H(f_{\mathrm{V}}(v_1)), \ldots, \mathrm{term}_H(f_{\mathrm{V}}(v_n))) \\
&= \mathrm{term}_H(\mathrm{res}(f_{\mathrm{E}}(e))) \\
&= \mathrm{term}_H(f_{\mathrm{V}}(\mathrm{res}(e))) \\
&= \mathrm{term}_H(f_{\mathrm{V}}(v)). \qquad \Box
\end{aligned}$$

The induction principle just used allows to show that a property $P$ holds for all nodes of a term graph. This principle is called *bottom-up induction* and is as follows:

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

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

**Definition 1.3.7 (Tree and fully collapsed term graph)**
A term graph $G$ is a *tree* if there is no $H$ with $G \prec H$, while $G$ is *fully collapsed* if there is no $H$ with $G \succ H$.

For example, the middle graph in Figure 1.5 is fully collapsed. The following characterization of trees and fully collapsed term graphs is easy to verify.

**Lemma 1.3.8**

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

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

The next lemma is proved in [91].

**Lemma 1.3.9**
*For every term graph $G$, there is a unique tree $\vartriangle G$ and a unique fully collapsed term graph $\triangledown G$ such that*

$$\vartriangle G \succeq G \succeq \triangledown G. \qquad \square$$

Hence, $\vartriangle G$ and $\triangledown G$ are the normal forms of $G$ with respect to $\prec$ and $\succ$. Note that by Lemma 1.3.6, $G$, $\vartriangle G$ and $\triangledown G$ represent the same term.

**Definition 1.3.10 (Bisimilarity)**
Two term graphs $G$ and $H$ are *bisimilar*, denoted by $G \sim H$, if $\mathrm{term}(G) = \mathrm{term}(H)$.

The three graphs in Figure 1.5, 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 [80]) 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 1.3.11 ([4])**
*For every term graph $G$, $\langle [G], \succeq \rangle$ is a complete lattice.* $\square$

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

**Corollary 1.3.12**
*For all term graphs $G$ and $H$, the following are equivalent:*

(1) $G \sim H$.

(2) $\vartriangle G = \vartriangle H$.

(3) $\triangledown G = \triangledown H$. $\square$

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

### *1.3.3 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 1.3.1, 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 [76,43,51,62,23,91].

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 [34,97,98,99], while [85,94,60,15,37,64, 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 [77,75,83,84].

In [36,4,2,68], 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].

## 1.4 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.

### *1.4.1 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 [54,10,28,67,86,57].

Let $\mathrm{T}_{\Sigma,\mathrm{X}}$ be the set of all terms over $\Sigma$ and X. A *substitution* is a mapping $\sigma\colon \mathrm{T}_{\Sigma,\mathrm{X}} \to \mathrm{T}_{\Sigma,\mathrm{X}}$ 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 $\langle l, r\rangle$ of terms, written $l \to r$, 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* $\langle \Sigma, \mathcal{R}\rangle$ consists of a set $\Sigma$ of function symbols and a set $\mathcal{R}$ of term rewrite rules over $\mathrm{T}_{\Sigma,\mathrm{X}}$. We will often identify such a system with its rule set $\mathcal{R}$, leaving $\Sigma$ implicit. A term rewriting system is *left-linear* (resp. *right-linear*) if all its rules are.

**Definition 1.4.1 (Term rewriting)**

The rewrite relation $\to$ on $\mathrm{T}_{\Sigma,\mathrm{X}}$ induced by a term rewriting system $\mathcal{R}$ is defined as follows: $t \to u$ if there is a rule $l \to r$ 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).

An *algebra* $A$ over $\Sigma$ consists of a non-empty set $\mathrm{D}_A$, distinguished elements $c_A \in \mathrm{D}_A$ for all constants $c$ in $\Sigma$, and $n$-ary functions $f_A\colon \mathrm{D}_A^n \to \mathrm{D}_A$ for all function symbols $f$ in $\Sigma$ with $\mathrm{arity}(f) = n \geq 1$. An *assignment* (or *valuation*) is a mapping $v\colon \mathrm{X} \to \mathrm{D}_A$, which is extended to a mapping $v\colon \mathrm{T}_{\Sigma,\mathrm{X}} \to \mathrm{D}_A$ 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 $v\colon \mathrm{X} \to \mathrm{D}_A$. 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 1.4.2 (Completeness of term rewriting)**
*For all terms $t$ and $u$,*

$$\mathcal{R} \models t \approx u \text{ if and only if } t \overset{*}{\leftrightarrow} u. \qquad \square$$

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 $\to$. In this case it suffices to rewrite $t$ and $u$ as long as possible, obtaining unique normal forms $t{\downarrow}$ and $u{\downarrow}$. Then $t \leftrightarrow^* u$ if and only if $t{\downarrow} = u{\downarrow}$.

### *1.4.2 Term Graph Rewriting*

In this subsection we define the application of term rewrite rules to term graphs. Given a rule $l \to r$ 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 1.4.3 ($\Diamond t$ and $\underline{\Diamond t}$)**
For every term $t$, let $\Diamond t$ be the term graph representing $t$ such that only variables are shared. That is, there is a graph morphism $f \colon \Delta t \to \Diamond t$ such that for all distinct edges $e_1$ and $e_2$,

$$f_{\mathrm{E}}(e_1) = f_{\mathrm{E}}(e_2) \text{ if and only if } \mathrm{lab}_{\Diamond t}(e_1) = \mathrm{lab}_{\Diamond t}(e_2) \in \mathrm{X}.$$

The graph resulting from $\Diamond t$ after removing all edges labelled with variables is denoted by $\underline{\Diamond t}$.

For example, Figure 1.6 shows the graphs $\Delta\mathtt{f(x,x)}$, $\Diamond\mathtt{f(x,x)}$ and $\underline{\Diamond\mathtt{f(x,x)}}$. Note that the latter graph is not a term graph according to Definition 1.3.1. It may be regarded as a term graph with an "open node". (In [15,91], such graphs are also regarded as term graphs, and term graphs without open nodes are said to be closed.)
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 1.4.4 (Instance and redex)**
A term graph $L$ is an *instance* of a term $l$ if there is graph morphism $\underline{\Diamond l} \to L$ sending $\mathrm{root}_{\Diamond l}$ to $\mathrm{root}_L$. Given a node $v$ in a term graph $G$ and a term rewrite rule $l \to r$, the pair $\langle v,\, l \to r\rangle$ is a *redex* if $G|_v$ is an instance of $l$.

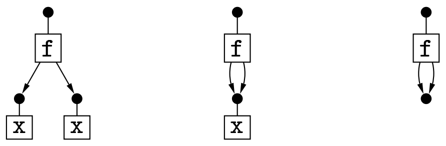


Figure 1.6: The graphs $\Delta$f(x,x), $\Diamond$f(x,x) and $\underline{\Diamond \mathtt{f(x,x)}}$

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

**Definition 1.4.5 (Term graph rewriting)**
Let $G$ be a term graph containing a redex $\langle v,\, l \to r\rangle$. Then there is a ***proper rewrite step*** $G \Rightarrow_{v,\, l\to r} H$, 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 $\mathrm{res}(e) = v$.

(2) $G_2$ is the graph obtained from the disjoint union $G_1 + \underline{\Diamond r}$ by

- identifying $v$ with $\mathrm{root}_{\Diamond r}$,
- identifying the image of $\mathrm{res}(e_1)$ with $\mathrm{res}(e_2)$, for each pair $\langle e_1, e_2\rangle \in \mathrm{E}_{\Diamond l} \times \mathrm{E}_{\Diamond r}$ with $\mathrm{lab}_{\Diamond l}(e_1) = \mathrm{lab}_{\Diamond r}(e_2) \in \mathrm{X}$.

(3) $H$ is the term graph obtained from $G_2$ by removing all nodes and edges not reachable from $\mathrm{root}_G$ ("garbage collection").[1]

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

**Example 1.1**
Figure 1.7 shows the three intermediate steps in the construction of a term graph rewrite step. The term rewrite rule applied to $G$ is $\mathtt{x} \times (\mathtt{y} + \mathtt{z}) \to (\mathtt{x} \times \mathtt{y}) + (\mathtt{x} \times \mathtt{z})$. In $G$ and $H$, shaded nodes and edges belong to the occurrences of $\underline{\Diamond \mathtt{x} \times (\mathtt{y} + \mathtt{z})}$ and $\underline{\Diamond (\mathtt{x} \times \mathtt{y}) + (\mathtt{x} \times \mathtt{z})}$, respectively. Note that the variables y and z correspond to the same node, that is, the graph morphism $\underline{\Diamond \mathtt{x} \times (\mathtt{y} + \mathtt{z})} \to G$ identifies the nodes representing y and z. □

The term graph rewrite relation $\Rightarrow$ is sound with respect to term rewriting in the sense that every proper step $G \Rightarrow_{v,\, l\to r} H$ corresponds to a sequence of

[1]More precisely, $H$ is the unique standard term graph isomorphic to the term graph constructed in this step.

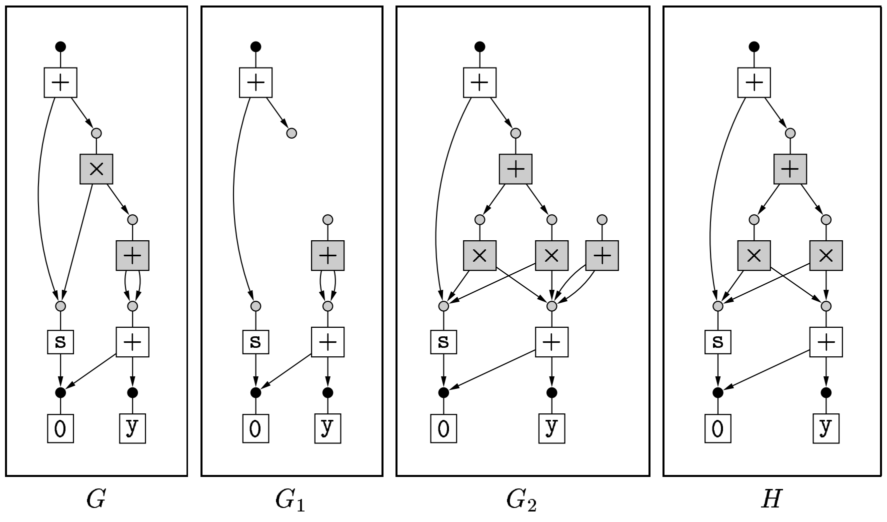


Figure 1.7: The intermediate steps in the construction of a term graph rewrite step

applications (or a parallel application) of $l \to r$ to several occurrences of the subterm $\mathrm{term}_G(v)$ in $\mathrm{term}(G)$. This explains the possible gain in time efficiency when passing from term rewriting to term graph rewriting.

**Theorem 1.4.6 (Soundness [50])**
*For all term graphs $G$ and $H$,*

$$G \underset{v}{\Rightarrow} H \textit{ implies } \mathrm{term}(G) \xrightarrow{+} \mathrm{term}(H).$$

*More precisely, we have $\mathrm{term}(G) \to^n \mathrm{term}(H)$, where $n$ is the number of paths from $\mathrm{root}_G$ to $v$.* □

A proof of this result can be found in [51,91].

**Example 1.2**
Consider the application of the rule $\mathtt{0} + \mathtt{x} \to \mathtt{x}$ shown in Figure 1.8. (Note that the graph $\underline{\Diamond\mathtt{x}}$ is a single node which is identified with the root of the redex and with the result node of the edge labelled with $\mathtt{s}$.) 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

$$((\mathtt{0} + \mathtt{s}(\mathtt{0})) \times (\mathtt{0} + \mathtt{s}(\mathtt{0}))) + (\mathtt{0} + \mathtt{s}(\mathtt{0})) \xrightarrow{3} (\mathtt{s}(\mathtt{0}) \times \mathtt{s}(\mathtt{0})) + \mathtt{s}(\mathtt{0}).$$ □

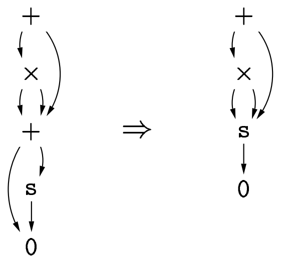


Figure 1.8: An application of the rule $0 + x \to x$

## 1.4.3 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 1.4.7 ($\Rightarrow_{\text{coll}}$, $\Rightarrow_{\text{copy}}$ and $\Rightarrow_{\text{bi}}$)**
The relations $\Rightarrow_{\text{coll}}$, $\Rightarrow_{\text{copy}}$ and $\Rightarrow_{\text{bi}}$ on term graphs are defined as follows:

$$\begin{aligned} \Rightarrow_{\text{coll}} &= \Rightarrow \cup \succ, \\ \Rightarrow_{\text{copy}} &= \Rightarrow \cup \prec, \\ \Rightarrow_{\text{bi}} &= \Rightarrow \cup \succ \cup \prec . \end{aligned}$$

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 1.4.6 if we replace $\to^+$ by $\to^*$. For, by Lemma 1.3.6, 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 $G \preceq \Delta G \succeq H$ (see Lemma 1.3.9 and Corollary 1.3.12).

**Example 1.3**

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

$$\begin{array}{rcl}\mathtt{fib(0)} & \to & \mathtt{0}\\ \mathtt{fib(s(0))} & \to & \mathtt{s(0)}\\ \mathtt{fib(s(s(x)))} & \to & \mathtt{fib(s(x))} + \mathtt{fib(x)}\end{array}$$

Using these three rules, evaluating a term of the form $\mathtt{fib(s}^n\mathtt{(0))}$ 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 $\mathtt{fib(s}^n\mathtt{(0))}$ 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 `fib`-redexes, the one representing the greater number is reduced. See Figure 1.9 for an illustration of this strategy. It is not difficult to verify that, for $n \geq 2$, this procedure evaluates $\mathtt{fib(s}^n\mathtt{(0))}$ 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 1.4.8**

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

$$G \overset{*}{\underset{\text{coll}}{\Rightarrow}} H \textit{ implies } G \overset{*}{\Rightarrow} H' \succeq H$$

*for some term graph $H'$.* □

Theorem 1.4.8 is a corollary of a result in [46] 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.

**Definition 1.4.9 (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.*

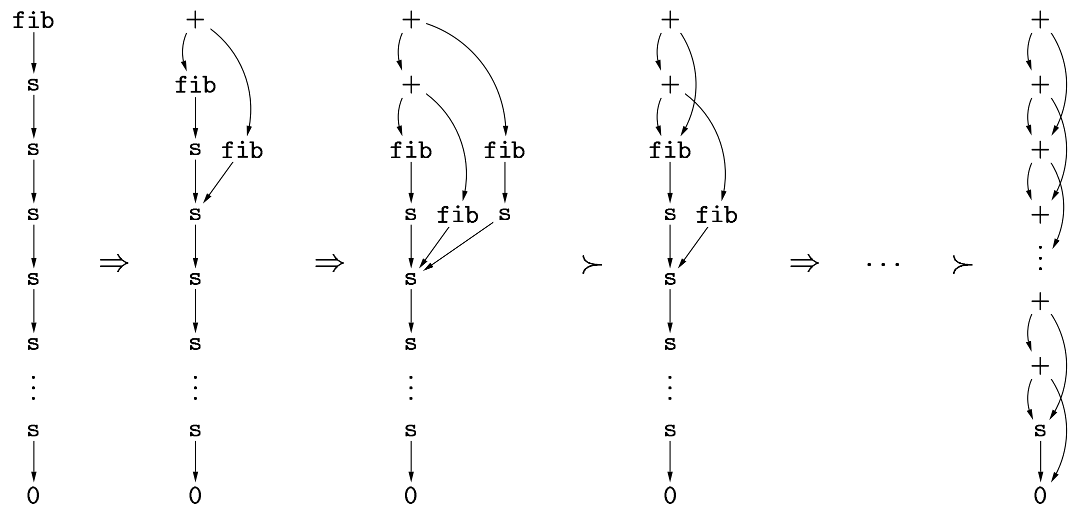


Figure 1.9: Collapsing to speed up evaluation

For example, in Figure 1.9 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 $H$, $\mathrm{term}(G) \to \mathrm{term}(H)$ implies $[G] \Rightarrow_{\sim} [H]$, and $[G] \Rightarrow_{\sim} [H]$ implies $\mathrm{term}(G) \to^{+} \mathrm{term}(H)$.

### *1.4.4 Bibliographic Notes*

Term graph rewriting was first studied in [97], 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 focusses on normalizing strategies and states the soundness of $\Rightarrow$ for left-linear rules.

In [15,64], 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 [15,64] 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 1.3.3) 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 1.4.5—so there is no garbage collection. Collapse steps (called *folding steps* in [43,51,91]) are also specified by suitable graph rewrite rules. The relation between jungle evaluation and the present setting is discussed in [91].

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.

## 1.5 Completeness

In this section we consider the completeness of term graph rewriting for simulating arbitrary term rewrite derivations (Subsection 1.5.1) and for computing term normal forms (Subsection 1.5.2). 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.

### *1.5.1 Simulating Arbitrary Term Rewrite Derivations*

From Theorem 1.4.6 we know that for every term graph rewrite derivation $G \Rightarrow^* H$ there is a corresponding term rewrite derivation $\text{term}(G) \to^* \text{term}(H)$. 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 1.4**

One obstacle to the completeness of plain term graph rewriting are non-left-linear term rewrite rules. For instance, the rule $\mathtt{eq(x,x)} \to \mathtt{true}$ cannot be applied to the tree $\triangle\mathtt{eq(0,0)}$ because there is no graph morphism $\underline{\Diamond\mathtt{eq(x,x)}} \to \triangle\mathtt{eq(0,0)}$ (see Figure 1.10). Hence, $\triangle\mathtt{eq(0,0)}$ is not reducible by $\Rightarrow$ or $\Rightarrow_{\text{copy}}$ although the represented term is reducible. Figure 1.10 also shows how to overcome the problem by collapsing: identifying the two occurrences of 0 enables a subsequent application of the rewrite rule. □

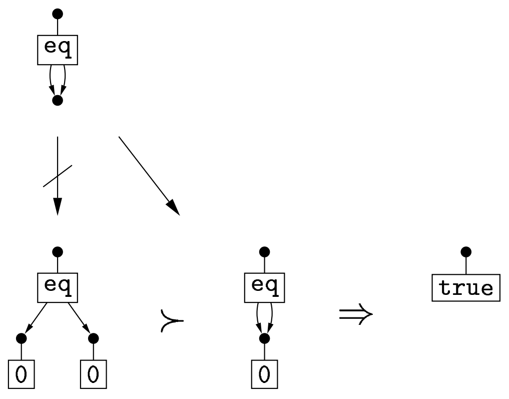


Figure 1.10: Collapsing to enable a rule application

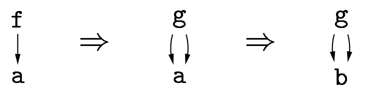


Figure 1.11: Applications of the rules $\mathtt{f(x)} \to \mathtt{g(x,x)}$ and $\mathtt{a} \to \mathtt{b}$

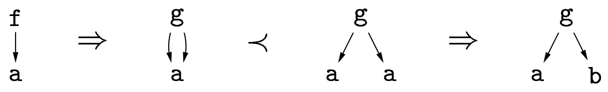


Figure 1.12: A $\Rightarrow_{\mathrm{copy}}$-derivation

**Example 1.5**
Even for left-linear systems, certain term rewrite derivations do not correspond to derivations by $\Rightarrow$ or $\Rightarrow_{\mathrm{coll}}$. Consider, for example, the rules $\mathtt{f(x)} \to \mathtt{g(x,x)}$ and $\mathtt{a} \to \mathtt{b}$. The derivation $\mathtt{f(a)} \to \mathtt{g(a,a)} \to \mathtt{g(a,b)}$ cannot be simulated by $\Rightarrow$ or $\Rightarrow_{\mathrm{coll}}$ because the application of the first rule leads to a shared constant a (see Figure 1.11). But this time we can simulate the given term rewrite derivation by $\Rightarrow_{\mathrm{copy}}$, as shown in Figure 1.12. □

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 1.5.1 (Simulation of term rewrite steps [90])**
*For every term rewrite step $t \to u$ there are term graphs $T$ and $U$ such that*

$$\Delta t \succeq T \Rightarrow U \preceq \Delta u. \qquad \Box$$

To illustrate Lemma 1.5.1, consider the rule $\mathtt{f(x+x)} \to \mathtt{f(x)+f(x)}$ which is neither left-linear nor right-linear. It admits the following term rewrite step:

$$\mathtt{g(f(0+0),0)} \to \mathtt{g(f(0)+f(0),0)}$$

Figure 1.13 shows how to simulate this step by term graph rewriting. In general, the subtree of $\Delta t$ corresponding to the replaced subterm $\sigma(l)$ 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 1.5.1, it is straightforward to show that every sequence of term rewrite steps can be simulated if both collapsing and copying are present.

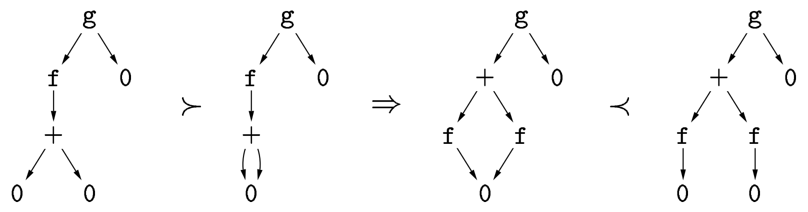


Figure 1.13: Simulation of a term rewrite step

**Theorem 1.5.2 (Completeness of $\Rightarrow_{\text{bi}}$ and $\Rightarrow_{\sim}$ [7,6])**
*For all term graphs $G$ and $H$, the following are equivalent:*

(1) $\text{term}(G) \to^* \text{term}(H)$.

(2) $G \Rightarrow^*_{\text{bi}} H$.

(3) $[G] \Rightarrow^*_{\sim} [H]$.

*Proof*
By the definitions of $\Rightarrow_{\text{bi}}$ 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 1.5.1, for every term rewrite step $t \to u$ there is a derivation $\Delta t \Rightarrow^+_{\text{bi}} \Delta u$. Hence, by induction on the length of derivations, $\text{term}(G) \to^* \text{term}(H)$ implies $\Delta\text{term}(G) \Rightarrow^*_{\text{bi}} \Delta\text{term}(H)$. Since $G \preceq \Delta\text{term}(G)$ and $\Delta\text{term}(H) \succeq H$, it follows $G \Rightarrow^*_{\text{bi}} H$. □

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

Combining the completeness of $\Rightarrow_{\text{bi}}$ with the observation that $\Leftrightarrow_{\text{bi}} = \Leftrightarrow_{\text{coll}} = \Leftrightarrow_{\text{copy}}$, we obtain the following corollary of Theorem 1.5.2.

**Corollary 1.5.3 (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 $\Leftrightarrow_{\text{coll}}$- respectively $\Leftrightarrow_{\text{copy}}$-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 1.4.2). Moreover, the decision procedure for equational validity described below Theorem 1.4.2 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 1.4 there is no conversion $\triangle\mathtt{eq}(0,0) \Leftrightarrow^* \triangle\mathtt{true}$ although $\mathtt{eq}(0,0) \approx \mathtt{true}$ is a valid equation, and in Example 1.5 there does not exist a conversion $\triangle\mathtt{f}(\mathtt{a}) \Leftrightarrow^* \triangle\mathtt{g}(\mathtt{a},\mathtt{b})$.

### *1.5.2 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 1.5.4 (Graph-reducibility)**
A term rewriting system $\mathcal{R}$ is *graph-reducible* by a binary relation $\Rrightarrow$ on term graphs if the following holds for every term graph $G$:

(1) For every normal form $H$ of $G$ with respect to $\Rrightarrow$, term$(H)$ is a normal form of term$(G)$ with respect to $\rightarrow$.

(2) If term$(G)$ has a normal form with respect to $\rightarrow$, then $G$ has a normal form with respect to $\Rrightarrow$.

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

Condition (1) ensures soundness of $\Rrightarrow$ 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 1.4 we already know that non-left-linear systems are not graph-reducible by $\Rightarrow$ and $\Rightarrow_{\text{copy}}$ in general: for $\mathcal{R} = \{\mathtt{eq}(\mathtt{x},\mathtt{x}) \rightarrow \mathtt{true}\}$, the tree $\triangle\mathtt{eq}(0,0)$ is a normal form with respect to $\Rightarrow$ and $\Rightarrow_{\text{copy}}$ although $\mathtt{eq}(0,0)$ is reducible by $\mathcal{R}$.

On the other hand, we will see that the system $\{\mathtt{f}(\mathtt{x}) \rightarrow \mathtt{g}(\mathtt{x},\mathtt{x}), \mathtt{a} \rightarrow \mathtt{b}\}$ of Example 1.5 is graph-reducible by $\Rightarrow$, $\Rightarrow_{\text{coll}}$ and $\Rightarrow_{\text{copy}}$.

**Definition 1.5.5 (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 $l_1 \to r_1$ and $l_2 \to r_2$ in $\mathcal{R}$, $l_1$ overlaps $l_2$ in a subterm $u$ only if $u = l_2$ and $(l_1 \to r_1) = (l_2 \to r_2)$. If $\mathcal{R}$ is non-overlapping and additionally left-linear, then it is an *orthogonal* system.

**Theorem 1.5.6** ([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 $\to$.

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

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 $\rightrightarrows$ on $\mathrm{T}_{\Sigma,\mathrm{X}}$ by modifying clause (2) of Definition 1.4.1 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 $t \rightrightarrows^* u$. The class of parallelly normalizing systems includes, for example, all orthogonal and all terminating term rewriting systems.

**Theorem 1.5.7** ([88])
*Every parallelly normalizing term rewriting system is graph-reducible by* $\Rightarrow_{\text{coll}}$. □

A term rewrite step $t \to u$ 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 [70]).

**Theorem 1.5.8** ([70])
*Every innermost normalizing term rewriting system is graph-reducible by* $\Rightarrow_{\text{coll}}$. □

We conclude this subsection by considering graph-reducubility by $\Rightarrow_{\mathrm{copy}}$. 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 $\Delta t \Rightarrow^*_{\mathrm{copy}} \Delta u$ (see [6]).

**Theorem 1.5.9**
*Every left-linear term rewriting system is strongly graph-reducible by* $\Rightarrow_{\mathrm{copy}}$. □

### *1.5.3 Bibliographic Notes*

Completeness of $\Rightarrow_{\mathrm{coll}}$ for proving equational validity was shown in [90]. 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 [64]. 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 [64] 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.

## 1.6 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 $\Rightarrow^+_{\mathrm{coll}}$ and the question whether $\Rightarrow_{\mathrm{coll}}$ is confluent are decidable then (provided $\mathcal{R}$ is finite). If $\Rightarrow_{\mathrm{coll}}$ 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 1.5.3).

### *1.6.1 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 $\Rightarrow_{\mathrm{coll}}$ is terminating. By restricting attention to right-linear systems, however, termination of $\Rightarrow_{\mathrm{coll}}$ 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 $\succ$ and $\prec$ are terminating relations.

**Theorem 1.6.1**

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

Note that this result does not hold for $\Rightarrow_{\mathrm{bi}}$, 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 1.6.1 also holds [6], while for $\Rightarrow_{\mathrm{copy}}$ one has to require that $\mathcal{R}$ is left-linear (otherwise the system $\{\mathtt{f(x,x)} \rightarrow \mathtt{f(a,a)}\}$ is a counterexample).

It is worth noting that by Theorem 1.6.1, 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 $\Rightarrow_{\mathrm{coll}}$ terminates "more often" than term rewriting.

**Example 1.6**

Consider the following two rules:

$$\begin{array}{rcl} \mathtt{f(a,b,x)} & \rightarrow & \mathtt{f(x,x,x)} \\ \mathtt{a} & \rightarrow & \mathtt{b} \end{array}$$

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

$$\mathtt{f(a,b,a)} \rightarrow \mathtt{f(a,a,a)} \rightarrow \mathtt{f(a,b,a)} \rightarrow \ldots$$

In contrast, $\Rightarrow$ and $\Rightarrow_{\mathrm{coll}}$ 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 $\mathtt{f}$-labelled edges the first two argument nodes of which are distinct, $n$ is the number of $\mathtt{a}$-labelled edges, and $p$ is the number of nodes in $G$. It is not difficult to check that for every step $G \Rightarrow_{\mathrm{coll}} H$, we have $\tau(G) > \tau(H)$. Thus, every sequence of $\Rightarrow_{\mathrm{coll}}$-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 1.4.8, if $\mathcal{R}$ is left-linear, then $\Rightarrow$ is terminating if and only if $\Rightarrow_{\mathrm{coll}}$ is terminating.

**Theorem 1.6.2**
*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 1.6.1. The "only if"-direction follows from the proof of Lemma 1.5.1. There, the collapsing $\triangle\, t \succeq T$ 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 $\underline{\Diamond r}$ is a tree (where $r$ is the right-hand side of the applied rewrite rule). It follows $U = \triangle u$. Hence every term rewrite sequence can be simulated by a sequence of $\Rightarrow_{\text{coll}}$-steps, which implies the proposition. □

**Corollary 1.6.3**
*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 1.6.2, termination of $\Rightarrow_{\text{coll}}$ cannot be decidable either. □

### 1.6.2 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 $\{\mathtt{a} \rightarrow \mathtt{b}\}$ and $\{\mathtt{b} \rightarrow \mathtt{a}\}$, yielding a non-terminating system. Even worse, Toyama [100] showed that the *disjoint* union of two terminating term rewriting systems need not be terminating. He gave the following counterexample.

**Example 1.7**
The two systems

$$\mathcal{R}_0 \left\{ \begin{array}{lcl} \mathtt{f(0,1,x)} & \to & \mathtt{f(x,x,x)} \end{array} \right.$$
$$\mathcal{R}_1 \left\{ \begin{array}{lcl} \mathtt{g(x,y)} & \to & \mathtt{x} \\ \mathtt{g(x,y)} & \to & \mathtt{y} \end{array} \right.$$

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

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

□

Toyama's observation stimulated several researchers to establish sufficient conditions under which the disjoint union of term rewriting systems preserves termination (see [41,82] 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 $\Rightarrow_{\mathrm{coll}}$ *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 $\Delta\mathtt{f(g(0,1),g(0,1),g(0,1))}$, one obtains the terminating derivations shown in Figure 1.14.

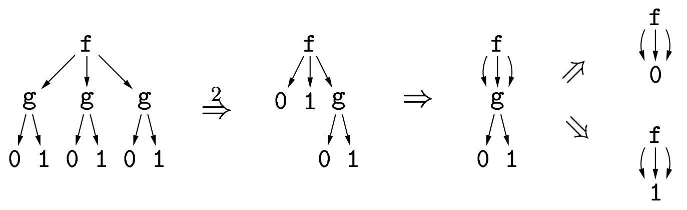


Figure 1.14: Two terminating derivations

The crucial point here is the application of the rule $\mathtt{f(0,1,x)} \to \mathtt{f(x,x,x)}$. In term rewriting, this rule produces three copies of the subterm $\mathtt{g(0,1)}$ which can be evaluated independently. In contrast, term graph rewriting yields a shared occurrence of this subterm, preventing that $\mathtt{f(0,1,x)} \to \mathtt{f(x,x,x)}$ can be applied again.

We will see that termination of $\Rightarrow_{\mathrm{coll}}$ for a composed system $\mathcal{R}_0 \cup \mathcal{R}_1$ can be guaranteed even when function symbols are shared between the left-hand sides respectively right-hand sides of $\mathcal{R}_0$ and $\mathcal{R}_1$.

**Definition 1.6.4 (Crosswise disjointness)**

Two term rewriting systems $\mathcal{R}_0$ and $\mathcal{R}_1$ are *crosswise disjoint* if the function symbols in the left-hand sides of the rules in $\mathcal{R}_i$ do not occur in the right-hand sides of the rules in $\mathcal{R}_{1-i}$, for $i = 0, 1$.

For example, the following systems are crosswise disjoint:

$$\mathcal{R}_0 \left\{ \begin{array}{rcl} \mathtt{f(x)} & \to & \mathtt{g(x,x)} \\ \mathtt{a} & \to & \mathtt{b} \end{array} \right.$$

$$\mathcal{R}_1 \left\{ \begin{array}{rcl} \mathtt{f(f(x))} & \to & \mathtt{g(x,b)} \\ \mathtt{h(a,x)} & \to & \mathtt{h(b,x)} \end{array} \right.$$

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

**Theorem 1.6.5** ([89])

*Let $\mathcal{R}_0 \cup \mathcal{R}_1$ be the union of two crosswise disjoint term rewriting systems. Then $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is terminating if and only if $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are terminating.* □

The main motivation for this result is to facilitate termination proofs. But we can also use it to sharpen Corollary 1.6.3, obtaining a stronger undecidability result for termination.

**Corollary 1.6.6**

*The following problem is undecidable in general:*

Instance: *A finite term rewriting system $\mathcal{R}$ such that $\to$ is not terminating.*
Question: *Is $\Rightarrow_{\mathcal{R}}$ terminating?*

*Proof*

If the above problem were decidable, we could decide termination of $\Rightarrow_{\mathcal{R}}$ for arbitrary finite systems $\mathcal{R}$ as follows. First construct the disjoint union $\mathcal{R}' = \mathcal{R} + \{\mathtt{f(a,b,x)} \to \mathtt{f(x,x,x)},\ \mathtt{a} \to \mathtt{b}\}$. By Example 1.6 and Theorem 1.6.5, $\mathcal{R}'$ is a non-terminating term rewriting system, and $\Rightarrow_{\mathcal{R}}$ is terminating if and only if $\Rightarrow_{\mathcal{R}'}$ is terminating. Thus, if the above problem were decidable, we could decide whether $\Rightarrow_{\mathcal{R}}$ is terminating or not. But this contradicts Corollary 1.6.3. □

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 1.6.7 (Constructor-sharing)**
Two term rewriting systems $\mathcal{R}_0$ and $\mathcal{R}_1$ are *constructor-sharing* if the defined symbols of $\mathcal{R}_i$ do not occur in $\mathcal{R}_{1-i}$, for $i = 0, 1$.

The proof of the following result was given in a framework based on terms with labels, but can be adapted to the present setting.

**Theorem 1.6.8 ([75])**
*Let $\mathcal{R}_0 \cup \mathcal{R}_1$ be the union of two constructor-sharing term rewriting systems. Then $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is terminating if and only if $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are terminating.* $\square$

An extended version of this result, also given in [75], additionally allows $\mathcal{R}_0$ and $\mathcal{R}_1$ to share defined symbols that do not occur in any right-hand side. In this form the result generalizes Theorem 1.6.5.

*Bibliographic Notes*

Theorem 1.6.5 was established in the framework of jungle evaluation, a proof for the present setting can be found in [91]. A short proof for the special case that $\mathcal{R}_0$ and $\mathcal{R}_1$ have disjoint function symbols is given in [84] (in an approach based on terms with labels).

A result even more general than Theorem 1.6.8 (and its extension) is presented in [72,73], but we refrain from stating it because of its technically involved premise. In [73] one can also find a condition—more general then disjointness—guaranteeing that normalization of $\Rightarrow_{\text{coll}}$ is preserved by combinations of systems.

### *1.6.3 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 [93], where a class of *simplification orders* on term graphs is established by extending Kruskal's Tree Theorem [74] from trees to term graphs.

**Definition 1.6.9 (Top and immediate subgraphs)**
Let $G$ be a term graph and $e$ be the unique edge such that $\text{att}_G(e) = \text{root}_G v_1 \dots v_n$ for some nodes $v_1, \dots, v_n$. Then the *top* of $G$, denoted by $\text{top}_G$, is the subgraph consisting of $e$ and the nodes $\text{root}_G, v_1, \dots, v_n$. The term graphs $G|_{v_1}, \dots, G|_{v_n}$ are the *immediate subgraphs* of $G$. We denote by $\text{Sub}_G$ the multiset $\{G|_{v_1}, \dots, G|_{v_n}\}$.

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 1.6.10 (Precedence)**
The set of all tops with function symbols from $\Sigma$ is denoted by $\text{Tops}_\Sigma$.[2] A *precedence* is a preorder $\sqsupseteq$ on $\text{Tops}_\Sigma$. The strict part and the equivalence part of $\sqsupseteq$ are defined by $\sqsupset = (\sqsupseteq - \sqsubseteq)$ and $\equiv = (\sqsupseteq \cap \sqsubseteq)$.

For example, Figure 1.15 shows a precedence over the function symbols of Example 1.6. To define the recursive path order, we recall from [29] the lifting

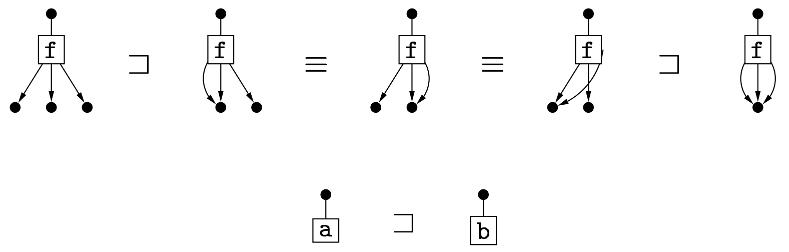


Figure 1.15: A precedence

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, $\mathcal{TG}_\Sigma$ denotes the set of all variable-free term graphs over $\Sigma$.

[2]Note that by our convention to deal with standard term graphs only, the tops in $\text{Tops}_\Sigma$ are pairwise non-isomorphic.

**Definition 1.6.11 (Recursive path order)**

Let $\sqsupseteq$ be a precedence. The *recursive path order* $>_{\mathrm{rpo}}$ on $\mathcal{TG}_\Sigma$ is defined inductively as follows: $G >_{\mathrm{rpo}} H$ if

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

(2) $\mathrm{top}_G \sqsupset \mathrm{top}_H$ and $G >_{\mathrm{rpo}} T$ for all immediate subgraphs $T$ of $H$, or

(3) $\mathrm{top}_G = \mathrm{top}_H$ and $\mathrm{Sub}_G >^{\mathrm{mul}}_{\mathrm{rpo}} \mathrm{Sub}_H$.

A more general variant of the recursive path order can be found in [93], where the equality $\mathrm{top}_G = \mathrm{top}_H$ in (3) is replaced by $\mathrm{top}_G \equiv \mathrm{top}_H$. The equality $S = H$ in (1) is relaxed similarly.

**Theorem 1.6.12**

*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 $\Rightarrow_{\mathrm{coll}}$, we have to consider precedences containing the collapsing of tops.

**Definition 1.6.13**

A precedence $\sqsupseteq$ is *collapse-compatible* if whenever there is a graph morphism $t \to u$ for some $t, u \in \mathrm{Tops}_\Sigma$, then $t \sqsupseteq u$. If moreover $\sqsupset$ is well-founded, then $\sqsupseteq$ is a *well-precedence.*

The precedence of Figure 1.15, for example, is a well-precedence. Recall, for the following theorem, that a term graph $L$ is an instance of a term $l$ if there is a root preserving graph morphism $\underline{\Diamond l} \to L$. A variable-free instance is called a *ground instance.*

**Theorem 1.6.14**

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

Using the precedence of Figure 1.15, we can convince ourselves that both rewrite rules of Example 1.6 satisfy the condition of Theorem 1.6.14. Thus, we obtain an alternative proof for the termination of $\Rightarrow_{\mathrm{coll}}$ over that system. We now give a further example for the use of the recursive path order.

**Example 1.8**
Consider the following rewrite system:

$$\begin{aligned} \neg\texttt{true} \wedge \texttt{false} &\rightarrow \neg\neg\neg\texttt{true} \\ \neg\neg\texttt{x} &\rightarrow \texttt{x} \wedge \texttt{x} \\ \neg\texttt{true} &\rightarrow \texttt{false} \end{aligned}$$

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

$$\neg\texttt{true} \wedge \texttt{false} \rightarrow \neg\neg\neg\texttt{true} \rightarrow \neg\texttt{true} \wedge \neg\texttt{true} \rightarrow \neg\texttt{true} \wedge \texttt{false} \rightarrow \ldots$$

However, termination of $\Rightarrow_{\text{coll}}$ can easily be checked by means of Theorem 1.6.14, using the well-precedence of Figure 1.16. □

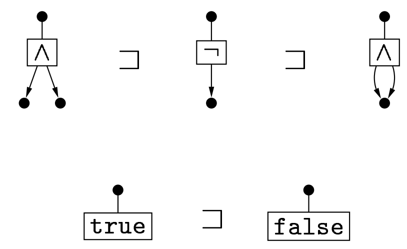


Figure 1.16: A well-precedence for Example 1.8

## 1.7 Confluence

An important consequence of the completeness of $\Rightarrow_{\text{coll}}$ for equational proofs (Corollary 1.5.3) 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.

### *1.7.1 The Relation to Term Rewriting*

We start with two counterexamples from [90] to demonstrate that confluence of term rewriting implies neither confluence of $\Rightarrow$ nor confluence of $\Rightarrow_{\text{coll}}$.

**Example 1.9**
Suppose that $\mathcal{R}$ is given as follows:[3]

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

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 1.17 shows that $\Rightarrow$ is neither normalizing nor confluent. The same applies to $\Rightarrow_{\text{coll}}$, as the collapse step $\Delta\mathtt{g(b,b)} \succ \nabla\mathtt{g(b,b)}$ does not essentially change the situation. The problem here is that the sharing created by the rule $\mathtt{f(x)} \to \mathtt{g(x,x)}$ prevents the rewrite step $\mathtt{g(a,a)} \to \mathtt{g(a,b)}$, which is necessary to reduce $\mathtt{g(a,a)}$ to $\mathtt{c}$. $\square$

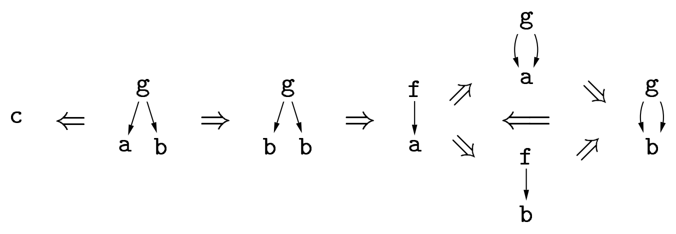


Figure 1.17: Non-confluence of $\Rightarrow$ and $\Rightarrow_{\text{coll}}$

**Example 1.10**
In the case of $\Rightarrow_{\text{coll}}$, confluence is not even guaranteed over an orthogonal one-rule system. Consider the rule

$$\mathtt{a} \to \mathtt{f(a)}$$

and suppose that $\Sigma$ contain a binary function symbol $\mathtt{g}$. Figure 1.18 shows two $\Rightarrow_{\text{coll}}$-derivations starting from $\Delta\mathtt{g(a,a)}$, where the resulting graphs do

[3]It is interesting to note that the same system was independently invented as a counterexample to completeness of (term-based) basic narrowing [78].

not have a common reduct. (The graphs derivable on the left represent the terms $\mathtt{g}(\mathtt{f}^n(\mathtt{a}), \mathtt{f}^n(\mathtt{a}))$, $n \geq 1$, while the graphs derivable on the right represent $\mathtt{g}(\mathtt{f}^n(\mathtt{a}), \mathtt{f}^{n+1}(\mathtt{a}))$, $n \geq 0$.) □

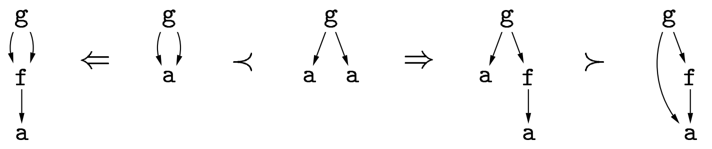


Figure 1.18: Non-confluence of $\Rightarrow_{\text{coll}}$

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 1.7.1 ([90])**
*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 $s \leftrightarrow^* u$, Corollary 1.5.3 gives $S \Leftrightarrow^*_{\text{coll}} U$. 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 1.2.4). 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 [91]. Note also that in Example 1.9, $\Rightarrow$ and $\Rightarrow_{\text{coll}}$ are not even locally confluent.

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 1.7.2**
*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 $\langle A, \to \rangle$ has *unique normal forms* if whenever $a \leftrightarrow^* b$ for normal forms $a$ and $b$, then $a = b$.

**Theorem 1.7.3** ([91])
*The relation $\Rightarrow_{\text{coll}}$ has unique normal forms if and only if $\to$ 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 $\triangledown t \Leftrightarrow^*_{\text{coll}} \triangledown u$ by completeness of $\Rightarrow_{\text{coll}}$. By Lemma 1.7.2, $\triangledown t$ and $\triangledown u$ are normal forms with respect to $\Rightarrow_{\text{coll}}$. Hence, by uniqueness of normal forms, $\triangledown t = \triangledown u$. With Corollary 1.3.12 follows $t = u$.
Conversely, suppose that $\to$ has unique normal forms. Let $G$ and $H$ be term graph normal forms such that $G \Leftrightarrow^*_{\text{coll}} H$. Then $\text{term}(G) \leftrightarrow^* \text{term}(H)$ by soundness of $\Rightarrow_{\text{coll}}$, and both terms are normal forms by Lemma 1.7.2. Thus, uniqueness of normal forms gives $\text{term}(G) = \text{term}(H)$. Hence $G = \triangledown\text{term}(G) = \triangledown\text{term}(H) = H$ by the uniqueness of fully collapsed term graphs (Lemma 1.3.9). □

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 1.7.4**

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

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

*Proof*
(1) The "only if"-direction is contained in Theorem 1.7.1. Conversely, let $\Rightarrow_{\text{coll}}$ be normalizing and $\to$ be confluent. The latter implies that $\to$ has unique normal forms (see Lemma 1.2.4). Hence, by Theorem 1.7.3, $\Rightarrow_{\text{coll}}$ 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 1.6.1, termination of $\to$ implies termination of $\Rightarrow_{\text{coll}}$. In particular, $\Rightarrow_{\text{coll}}$ is normalizing then. Hence, by (1), confluence carries over from $\to$ to $\Rightarrow_{\text{coll}}$. □

The next example shows that the converse of the second part of Corollary 1.7.4 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 1.11**
Consider the following system:

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

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

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

Using the recursive path order induced by the well-precedence of Figure 1.19, it is not difficult to see that Theorem 1.6.14 ensures termination of $\Rightarrow_{\text{coll}}$. Then Corollary 1.7.4(1) guarantees that $\Rightarrow_{\text{coll}}$ is confluent, and hence convergent. □

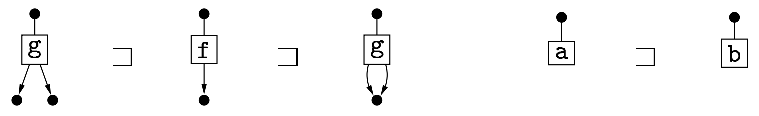


Figure 1.19: A well-precedence for Example 1.11

### *1.7.2 Decidability and Combined Systems*

Analogously to the situation for term rewriting [69], 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 [91,92].

**Theorem 1.7.5** ([91])
*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 1.7.4(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 [101], 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 1.10, where $\Rightarrow_{\text{coll}}$ is confluent when $\Sigma = \{\mathtt{a}, \mathtt{f}\}$. After adding a binary symbol $\mathtt{g}$, 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 1.7.6 (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 1.6.2, we write $\Rightarrow_{\mathcal{R}}$ for the relation $\Rightarrow_{\text{coll}}$ over a term rewriting system $\mathcal{R}$.

**Theorem 1.7.7** ([90])

*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 [73]. 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 1.12**

Consider the following two systems with disjoint function symbols:

$$\mathcal{R}_0 \left\{ \begin{array}{rcl} \mathtt{f(0,1,x)} & \to & \mathtt{f(x,x,x)} \\ \mathtt{f(x,y,z)} & \to & \mathtt{2} \\ \mathtt{0} & \to & \mathtt{2} \\ \mathtt{1} & \to & \mathtt{2} \end{array} \right.$$

$$\mathcal{R}_1 \left\{ \begin{array}{rcl} \mathtt{g(x,y,y)} & \to & \mathtt{x} \\ \mathtt{g(x,x,y)} & \to & \mathtt{y} \end{array} \right.$$

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:

$$\begin{array}{rcl} \mathtt{f(g(0,1,1),g(0,1,1),g(0,1,1))} & \to & \mathtt{f(0,g(0,1,1),g(0,1,1))} \\ & \to^2 & \mathtt{f(0,g(2,2,1),g(0,1,1))} \\ & \to & \mathtt{f(0,1,g(0,1,1))} \\ & \to & \mathtt{f(g(0,1,1),g(0,1,1),g(0,1,1))} \\ & \to & \ldots \end{array}$$

Theorem 1.7.7 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 1.7.4(2). In particular, every exhaustive rewrite sequence starting from $\Delta\mathtt{f}(\mathtt{g}(0,1,1),\mathtt{g}(0,1,1),\mathtt{g}(0,1,1))$ ends in the unique normal form $\Delta 2$. □

### *1.7.3 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 $\Rightarrow_{\mathrm{bi}}$ and $\Rightarrow_{\sim}$. (For the first two theorems, the reader may wish to look up Definition 1.5.5 which introduces orthogonal systems.)

**Theorem 1.7.8 ([97])**
*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 1.13**
The left-linear system

$$\begin{array}{rcl} \mathtt{f}(\mathtt{x}) & \to & \mathtt{g}(\mathtt{x},\mathtt{x}) \\ \mathtt{f}(\mathtt{a}) & \to & \mathtt{g}(\mathtt{a},\mathtt{a}) \end{array}$$

is clearly convergent under term rewriting, but Figure 1.20 shows that plain term graph rewriting is not confluent. □

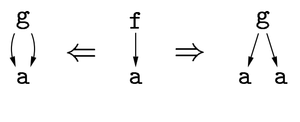


Figure 1.20: Non-confluence of $\Rightarrow$

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 1.7.9 (Confluence modulo bisimilarity)**
A binary relation $\Rrightarrow$ on term graphs is *confluent modulo bisimilarity* if whenever $G_1 \Lleftarrow^* G \sim H \Rrightarrow^* H_1$, there are term graphs $G_2$ and $H_2$ such that $G_1 \Rrightarrow^* G_2 \sim H_2 \Lleftarrow^* H_1$.

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

**Theorem 1.7.10 ([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 1.7.11 ([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 1.9, 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 $\Rightarrow_{\mathrm{bi}}$ and $\Rightarrow_{\sim}$ are confluent exactly for all confluent term rewriting systems.

**Theorem 1.7.12 ([7,6])**
*The following are equivalent:*

(1) $\Rightarrow_{\mathrm{bi}}$ *is confluent.*
(2) $\Rightarrow_{\sim}$ *is confluent.*
(3) $\rightarrow$ *is confluent.* □

We finally mention that for $\Rightarrow_{\mathrm{coll}}$, $\Rightarrow_{\mathrm{copy}}$ and $\Rightarrow_{\mathrm{bi}}$, 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]).

## 1.8 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 1.8.2 we discuss the completeness of two restricted forms of term graph narrowing, called *minimally collapsing* and *maximally collapsing* narrowing. Our presentation is based on [44,46].

### *1.8.1 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 1.8.1 (Term graph substitution)**
A *term graph substitution* is a finite set $\alpha = \{x_1/G_1, \dots, x_n/G_n\}$ of substitution pairs such that $x_1, \dots, x_n$ are pairwise distinct and $x_i \neq \text{term}(G_i)$ for $i = 1, \dots, n$. Given a term graph $H$, applying $x_1/G_1, \dots, x_n/G_n$ simultaneously to all edges labelled with $x_1, \dots, x_n$ yields the term graph $H\alpha$.

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

$$\alpha\beta = \{x/G\beta \mid x/G \in \alpha \text{ and } x \neq \text{term}(G\beta)\} \cup \{y/H \in \beta \mid y \notin \text{Dom}(\alpha)\}$$

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

A term graph substitution $\alpha$ induces the term substitution $\alpha^{\text{term}}\colon \mathrm{T}_{\Sigma,\mathrm{X}} \to \mathrm{T}_{\Sigma,\mathrm{X}}$ mapping $x_i$ to $\text{term}(G_i)$, for $i = 1, \dots, n$, 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 $\{x_1/t_1, \dots, x_n/t_n\}$ if $x_i\sigma = t_i$ for $i = 1, \dots, n$ and $x\sigma = x$ for each other variable $x$.

A *variant* of a term rewrite rule $l \to r$ is a rule of the form $l\sigma \to r\sigma$, where $\sigma$ is an injective substitution mapping variables to variables. A set of terms $\{t_1, \dots, t_n\}$ 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 1.8.2 (Term graph narrowing)**
Let $G$ and $H$ be term graphs, $U$ a set of non-variable nodes in $G$, $l \to r$ a variant of a rule in $\mathcal{R}$, and $\alpha$ a term graph substitution. There is a *narrowing step* $G \rightsquigarrow_{U,\, l\to r,\, \alpha} H$ if $\alpha^{\text{term}}$ is a most general unifier of $\{\text{term}_G(u) \mid u \in U\} \cup \{l\}$, and

$$G\alpha \succeq G' \underset{v,\, l\to r}{\Longrightarrow} H$$

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

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

From now on we assume that $\mathcal{R}$ contains the rule $\mathtt{x} =^? \mathtt{x} \to \mathtt{true}$, where the binary symbol $=^?$ and the constant $\mathtt{true}$ 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 $\mathtt{true}$. 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 1.14**
Let $\mathcal{R}$ consist of the following rules:

$$\begin{array}{rcl}
\mathtt{0} + \mathtt{x} & \to & \mathtt{x} \\
\mathtt{s(x)} + \mathtt{y} & \to & \mathtt{s(x + y)} \\
\mathtt{0} \times \mathtt{x} & \to & \mathtt{0} \\
\mathtt{s(x)} \times \mathtt{y} & \to & (\mathtt{x} \times \mathtt{y}) + \mathtt{y} \\
\mathtt{x} =^? \mathtt{x} & \to & \mathtt{true}
\end{array}$$

Suppose that we want to solve the goal $(\mathtt{z} \times \mathtt{z}) + (\mathtt{z} \times \mathtt{z}) =^? \mathtt{s(z)}$. Figure 1.21 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 1.8.2 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 $\{\mathtt{x/0},\ \mathtt{x'/s(0)},\ \mathtt{y/s(0)},\ \mathtt{z/s(0)}\}$. Restricting this substitution to the variables of the goal yields the solution $\{\mathtt{z/s(0)}\}$. 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 $V \subseteq \mathrm{X}$, we write $\sigma =_{\mathcal{R}} \tau\ [V]$ if $x\sigma \leftrightarrow^* x\tau$ for each $x \in V$, and $\sigma \leq_{\mathcal{R}} \tau\ [V]$ if there is a substitution $\rho$ such that $\sigma\rho =_{\mathcal{R}} \tau\ [V]$. The set of variables occurring in a term graph $G$ is denoted by $\mathrm{Var}(G)$, that is, $\mathrm{Var}(G) = \mathrm{lab}_G(\mathrm{E}_G) \cap \mathrm{X}$.

**Theorem 1.8.3 (Soundness and completeness of narrowing)**
*Let $G$ be a term graph such that* $\mathrm{term}(G)$ *is a goal $s =^? t$.*

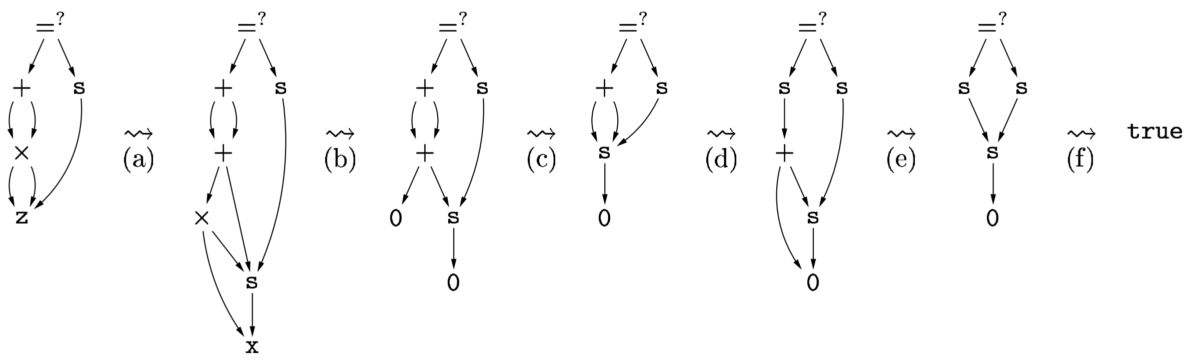
(1) *If $G \rightsquigarrow_\alpha^* \Delta\mathtt{true}$, then $\alpha^{\mathrm{term}}$ is a solution of $s =^? t$.*
(2) *If $\Rightarrow_{\mathrm{coll}}$ is normalizing and confluent, then for every solution $\sigma$ of $s =^? t$ there exists a narrowing derivation $G \rightsquigarrow_\beta^* \Delta\mathtt{true}$ such that $\beta^{\mathrm{term}} \leq_{\mathcal{R}} \sigma\ [\mathrm{Var}(G)]$.* □

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

**Example 1.15**
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 [79]. As a counterexample we can use the system of Example 1.9:

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

After adding the rule $\mathtt{x} =^? \mathtt{x} \to \mathtt{true}$, term rewriting remains normalizing and confluent. Since $\mathtt{f(a)}$ is reducible to $\mathtt{c}$, the empty substitution is a solution of



| step | rewrite rule | substitution |
|---|---|---|
| (a) | $\mathtt{s(x)} \times \mathtt{y} \to (\mathtt{x} \times \mathtt{y}) + \mathtt{y}$ | $\{\mathtt{y/s(x)},\ \mathtt{z/s(x)}\}$ |
| (b) | $\mathtt{0} \times \mathtt{x}' \to \mathtt{0}$ | $\{\mathtt{x/0},\ \mathtt{x}'\mathtt{/s(0)}\}$ |
| (c) | $\mathtt{0} + \mathtt{x} \to \mathtt{x}$ | $\{\mathtt{x/s(0)}\}$ |
| (d) | $\mathtt{s(x)} + \mathtt{y} \to \mathtt{s(x + y)}$ | $\{\mathtt{x/0},\ \mathtt{y/s(0)}\}$ |
| (e) | $\mathtt{0} + \mathtt{x} \to \mathtt{x}$ | $\{\mathtt{x/s(0)}\}$ |
| (f) | $\mathtt{x} =^? \mathtt{x} \to \mathtt{true}$ | $\{\mathtt{x/s(s(0))}\}$ |

Figure 1.21: A term graph narrowing derivation

the goal f(a) $=^?$ c . But Figure 1.22 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, $\Rightarrow_{\mathrm{coll}}$ is neither normalizing nor confluent (see Figure 1.17 for the absense of confluence). □

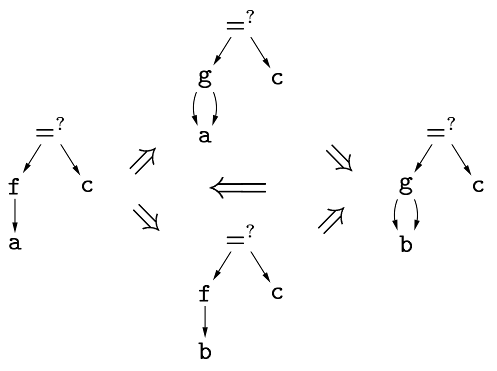


Figure 1.22: Incompleteness of term graph narrowing

### *1.8.2 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 1.8.4 (Minimal collapsing)**
A collapsing $G \succeq M$ is *minimal* with respect to a redex $\langle v,\, l \to r\rangle$ in $M$ if for each term graph $M'$ with $G \succeq M' \succ M$ and each preimage $v'$ of $v$ in $M'$, the pair $\langle v',\, l \to r\rangle$ is not a redex.

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

**Definition 1.8.5 (Minimally collapsing narrowing)**
A term graph narrowing derivation is *minimally collapsing* if for each narrowing step $G \mapsto G\alpha \succeq G' \Rightarrow_{v,\,l\to r} H$, the collapsing $G\alpha \succeq G'$ is minimal with respect to the redex $\langle v,\, l \to r\rangle$.

For example, the derivation of Figure 1.21 is minimally collapsing. Note that in a minimally collapsing step $G \rightsquigarrow_{U,\, l \to r,\, \alpha} H$, the set $U$ must be a singleton. It turns out that Theorem 1.8.3 can be strengthened by replacing unrestricted term graph narrowing with minimally collapsing narrowing.

**Theorem 1.8.6 (Completeness of minimally collapsing narrowing)**
*Minimally collapsing narrowing is complete whenever $\Rightarrow_{\mathrm{coll}}$ 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 1.8.7 (Maximally collapsing narrowing)**
A term graph narrowing derivation is *maximally collapsing* if for each narrowing step $G \mapsto G\alpha \succeq G' \Rightarrow_{v,\, l \to r} H$, the term graph $G'$ is fully collapsed.

**Example 1.16**
Consider the rules

$$\begin{array}{rcl} \mathtt{exp(0)} & \to & \mathtt{s(0)} \\ \mathtt{exp(s(x))} & \to & \mathtt{exp(x)} + \mathtt{exp(x)} \end{array}$$

specifying the function $\exp\colon n \mapsto 2^n$ on natural numbers. Figure 1.23 demonstrates that maximally collapsing narrowing can solve a goal of the form

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

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.

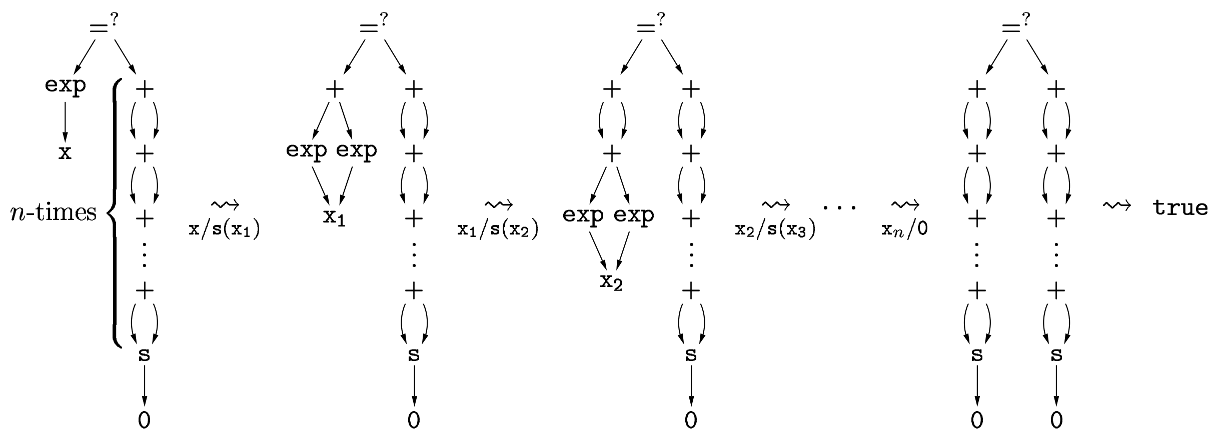


Figure 1.23: A maximally collapsing narrowing derivation

**Example 1.17**
Consider the following term rewriting system:

$$\begin{array}{rcl} \mathtt{g(x,y)} & \to & \mathtt{g(a,a)} \\ \mathtt{a} & \to & \mathtt{b} \\ \mathtt{g(a,b)} & \to & \mathtt{b} \\ \mathtt{x} =^? \mathtt{x} & \to & \mathtt{true} \end{array}$$

Here $\Rightarrow_{\text{coll}}$ is normalizing and confluent, which can be shown by induction on the size of term graphs. But the tree $\Delta\mathtt{g(a,a)}$ cannot be reduced to its normal form $\Delta\mathtt{b}$ if proper rewrite steps are preceded by maximal collapse steps, as shown in Figure 1.24. It follows that maximally collapsing narrowing cannot solve the goal $\mathtt{g(a,a)} =^? \mathtt{b}$, although $\mathtt{g(a,a)}$ and $\mathtt{b}$ are clearly equivalent. □

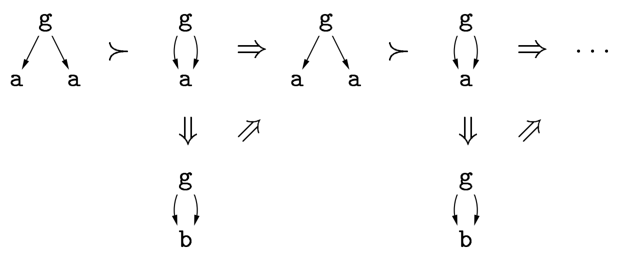


Figure 1.24: Incompleteness of maximally collapsing narrowing

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

**Theorem 1.8.8 (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.

### *1.8.3 Bibliographic Notes*

The present definition of term graph narrowing was introduced in [44]. It extends the definition in [46] in that the latter corresponds to the special case where all nodes in the set $U$ represent the same term. The paper [44] 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 1.8.8 holds for maximally collapsing basic narrowing as well.

Basic narrowing on term graphs is also addressed in [71], but the kind of narrowing used there (going back to [45]) 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 $\{\mathtt{f(x,x)} \to \mathtt{a}\}$ (consider, for example, the goal $\mathtt{f(x,y)} =^? \mathtt{a}$).

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.

## 1.9 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 [97,98,99,77]. 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 [40,66], while [61,39,13] deal with the term graph rewrite language Dactl. A report on an implementation which enforces full collapsing can be found in [59]. For a system dealing with term graphs with bound variables see [58].

Aspects of term graph rewriting relevant for the design, implementation and analysis of functional programming languages are discussed in [87,3,2]. See also the next chapter and the references given there. 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 [36,37,64,65,18,20].

Further issues that have been considered are term graph rewriting over conditional term rewriting systems [83], the relation between term graph rewriting and event structures [63,17], 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

1. Harold Abelson and Gerald Jay Sussman. *Structure and Interpretation of Computer Programs.* The MIT Press, 1985.
2. Zena M. Ariola. Relating graph and term rewriting via Böhm models. *Applicable Algebra in Engineering, Communication and Computing*, 7:401–426, 1996.
3. Zena M. Ariola and Arvind. Properties of a first-order functional language with sharing. *Theoretical Computer Science*, 146:69–108, 1995.
4. Zena M. Ariola and Jan Willem Klop. Equational term graph rewriting. *Fundamenta Informaticae*, 26:207–240, 1996.
5. Zena M. Ariola and Jan Willem Klop. Lambda calculus with explicit recursion. *Information and Computation*, 139:154–233, 1997.
6. Zena M. Ariola, Jan Willem Klop, and Detlef Plump. Bisimilarity in term graph rewriting. *Information and Computation.* To appear.
7. Zena M. Ariola, Jan Willem Klop, and Detlef Plump. Confluent rewriting of bisimilar term graphs. In *Proc. Fourth Workshop on Expressiveness in Concurrency*, volume 7 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1997.
8. Andrea Asperti and Cosimo Laneve. Interaction systems I: The theory of optimal reductions. *Mathematical Structures in Computer Science*, 4:457–504, 1994.
9. Jürgen Avenhaus. *Reduktionssysteme.* Springer-Verlag, 1995.
10. Jürgen Avenhaus and Klaus Madlener. Term rewriting and equational reasoning. In R.B. Banerij, editor, *Formal Techniques in Artificial Intelligence: a Sourcebook*, pages 1–43. Elsevier, 1990.
11. Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

12. Richard Banach. Term graph rewriting and garbage collection using opfibrations. *Theoretical Computer Science*, 131:29–94, 1994.
13. Richard Banach. Fundamental issues and the design of MONSTR. *Journal of Universal Computer Science*, 2:164–216, 1996.
14. Richard Banach. Transitive term graph rewriting. *Information Processing Letters*, 60:109–114, 1996.
15. Hendrik Barendregt, Marko van Eekelen, John Glauert, Richard Kennaway, Rinus Plasmeijer, and Ronan Sleep. Term graph rewriting. In *Proc. Parallel Architectures and Languages Europe*, volume 259 of *Lecture Notes in Computer Science*, pages 141–158. Springer-Verlag, 1987.
16. Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Texts and Monographs in Computer Science. Springer-Verlag, 1993.
17. David Clark and Richard Kennaway. Event structures and non-orthogonal term graph rewriting. *Mathematical Structures in Computer Science*, 6(6):545–578, 1996.
18. Andrea Corradini and Frank Drewes. (Cyclic) term graph rewriting is adequate for rational parallel term rewriting. Technical Report TR-97-14, Universitá di Pisa, Dipartimento di Informatica, 1997.
19. Andrea Corradini and Fabio Gadducci. A 2-categorical presentation of term graph rewriting. In *Proc. Category Theory and Computer Science*, volume 1290 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.
20. Andrea Corradini and Fabio Gadducci. Rational term rewriting. In *Proc. Foundations of Software Science and Computation Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 156–171. Springer-Verlag, 1998.
21. Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic approaches to graph transformation — Part I: Basic concepts and double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume I, chapter 3, pages 163–245. World Scientific, 1997.
22. Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, and Michael Löwe. Graph grammars and logic programming. In *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 221–237. Springer-Verlag, 1991.
23. Andrea Corradini and Francesca Rossi. Hyperedge replacement jungle rewriting for term rewriting systems and logic programming. *Theoretical Computer Science*, 109:7–48, 1993.

24. Andrea Corradini, Francesca Rossi, and Francesco Parisi-Presicce. Logic programming as hypergraph rewriting. In *Proc. CAAP '91*, volume 493 of *Lecture Notes in Computer Science*, pages 275–295. Springer-Verlag, 1991.
25. Andrea Corradini and Dietmar Wolz. Jungle rewriting: an abstract description of a lazy narrowing machine. In *Proc. Graph Transformations in Computer Science*, volume 776 of *Lecture Notes in Computer Science*, pages 119–137. Springer-Verlag, 1994.
26. Nachum Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
27. Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.
28. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science,* volume B, chapter 6, pages 244–320. Elsevier, 1990.
29. Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.
30. Peter J. Downey, Ravi Sethi, and Robert E. Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758–771, 1980.
31. Klaus Drosten. *Termersetzungssysteme.* Informatik-Fachberichte 210. Springer-Verlag, 1989.
32. Rachid Echahed and Jean-Christophe Janodet. Admissible graph rewriting and narrowing. In *Proc. Joint International Conference and Symposium on Logic Programming*, pages 325–342. MIT Press, 1998.
33. Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In *Proc. Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 1–69. Springer-Verlag, 1979.
34. Hartmut Ehrig and Barry K. Rosen. Commutativity of independent transformations on complex objects. Research Report RC 6251, IBM T.J. Watson Research Center, Yorktown Heights, 1976.
35. Joost Engelfriet and Linda Heyker. Context-free hypergraph grammars have the same term-generating power as attribute grammars. *Acta Informatica*, 29:161–210, 1992.
36. William M. Farmer, John D. Ramsdell, and Ronald J. Watro. A correctness proof for combinator reduction with cycles. *ACM Transactions on Programming Languages and Systems*, 12:123–134, 1990.
37. William M. Farmer and Ronald J. Watro. Redex capturing in term graph rewriting. *International Journal of Foundations of Computer Science*,

1(4):369–386, 1990.

38. Philippe Flajolet, Paolo Sipala, and Jean-Marc Steyaert. Analytic variations on the common subexpression problem. In *Proc. Automata, Languages, and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer-Verlag, 1990.
39. John Glauert, Richard Kennaway, and Ronan Sleep. Dactl: An experimental graph rewriting language. In *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 378–395. Springer-Verlag, 1991.
40. Joseph Goguen, Claude Kirchner, and José Meseguer. Concurrent term rewriting as a model of computation. In *Proc. Graph Reduction*, volume 279 of *Lecture Notes in Computer Science*, pages 53–93. Springer-Verlag, 1987.
41. Bernhard Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.
42. Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump. Jungle evaluation. In *Proc. Recent Trends in Data Type Specification*, volume 332 of *Lecture Notes in Computer Science*, pages 92–112. Springer-Verlag, 1988.
43. Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump. Jungle evaluation. *Fundamenta Informaticae*, 15(1):37–60, 1991.
44. Annegret Habel and Detlef Plump. Complete strategies for term graph narrowing. In *Recent Developments in Algebraic Development Techniques, Selected Papers*, Lecture Notes in Computer Science. Springer-Verlag. To appear.
45. Annegret Habel and Detlef Plump. Unification, rewriting, and narrowing on term graphs. In *Proc. Joint* COMPUGRAPH/SEMAGRAPH *Workshop on Graph Rewriting and Computation (SEGRAGRA '95)*, volume 2 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1995.
46. Annegret Habel and Detlef Plump. Term graph narrowing. *Mathematical Structures in Computer Science*, 6:649–676, 1996.
47. Gaétan Hains. The compaction of acyclic terms. In *Proc. Parallel Architectures and Languages Europe*, volume 366 of *Lecture Notes in Computer Science*, pages 288–303. Springer-Verlag, 1989.
48. Michael Hanus. The integration of functions into logic programming: From theory to practice. *The Journal of Logic Programming*, 19 & 20:583–628, 1994.
49. Berthold Hoffmann. Term rewriting with sharing and memoïzation. In *Proc. Algebraic and Logic Programming*, volume 632 of *Lecture Notes in Computer Science*, pages 128–142. Springer-Verlag, 1992.

50. Berthold Hoffmann and Detlef Plump. Jungle evaluation for efficient term rewriting. In *Proc. Algebraic and Logic Programming.* Mathematical Research 49, pages 191–203, Berlin, 1988. Akademie-Verlag. Also in Springer Lecture Notes in Computer Science 343, 191–203, 1989.
51. Berthold Hoffmann and Detlef Plump. Implementing term rewriting by jungle evaluation. *RAIRO Theoretical Informatics and Applications*, 25(5):445–472, 1991.
52. Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
53. Gérard Huet and Dallas Lankford. On the uniform halting problem for term rewriting systems. Report no. 283, INRIA Rocquencourt, 1978.
54. Gérard Huet and Derek C. Oppen. Equations and rewrite rules, a survey. In Ronald V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, 1980.
55. Jean-Marie Hullot. Canonical forms and unification. In *Proc. 5th International Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, 1980.
56. Matthias Jantzen. *Confluent String Rewriting*, volume 14 of *EATCS Monographs.* Springer-Verlag, 1988.
57. Matthias Jantzen. Basics of term rewriting. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages,* volume 3, chapter 5, pages 269–337. Springer-Verlag, 1997.
58. Wolfram Kahl. Internally typed second-order term graphs. In *Proc. Graph-Theoretic Concepts in Computer Science*, volume 1517 of *Lecture Notes in Computer Science*, pages 149–163. Springer-Verlag, 1998.
59. Stefan Kahrs. Unlimp: Uniqueness as a leitmotiv for implementation. In *Proc. Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 115–129. Springer-Verlag, 1992.
60. Richard Kennaway. On "On graph rewritings". *Theoretical Computer Science*, 52:37–58, 1987. Corrigendum: *Theoretical Computer Science*, 61:317–320, 1988.
61. Richard Kennaway. Implementing term rewrite languages in Dactl. *Theoretical Computer Science*, 72:225–249, 1990.
62. Richard Kennaway. Graph rewriting in some categories of partial morphisms. In *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 490–504. Springer-Verlag, 1991.
63. Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. Event structures and orthogonal term graph rewriting. In *Term Graph*

*Rewriting: Theory and Practice*, chapter 11, pages 141–169. John Wiley, 1993.

64. Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. On the adequacy of graph rewriting for simulating term rewriting. *ACM Transactions on Programming Languages and Systems*, 16(3):493–523, 1994.
65. Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. Transfinite reductions in orthogonal term rewriting systems. *Information and Computation*, 119:18–38, 1995.
66. Claude Kirchner and Patrick Viry. Implementing parallel rewriting. In *Proc. Programming Language Implementation and Logic Programming*, volume 456 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1990.
67. Jan Willem Klop. Term rewriting systems. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science,* volume 2, pages 1–116. Oxford University Press, 1992.
68. Jan Willem Klop. Term graph rewriting. In *Proc. Higher-Order Algebra, Logic, and Term Rewriting*, volume 1074 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1996.
69. Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebras*, pages 263–297. Pergamon Press, 1970.
70. Madala R.K. Krishna Rao. Graph reducibility of term rewriting systems. In *Proc. Mathematical Foundations of Computer Science 1995*, volume 969 of *Lecture Notes in Computer Science*, pages 371–381. Springer-Verlag, 1995.
71. Madala R.K. Krishna Rao. Completeness results for basic narrowing in non-copying implementations. In *Proc. Joint International Conference and Symposium on Logic Programming*, pages 393–407. MIT Press, 1996.
72. Madala R.K. Krishna Rao. Modularity of termination in term graph rewriting. In *Proc. Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 230–244. Springer-Verlag, 1996.
73. Madala R.K. Krishna Rao. Modular aspects of term graph rewriting. *Theoretical Computer Science*, 208(1-2):59–86, 1998.
74. Joseph B. Kruskal. Well-quasi-ordering, the Tree Theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960.
75. Masahito Kurihara and Azuma Ohuchi. Modularity in noncopying term rewriting. *Theoretical Computer Science*, 152(1):139–169, 1995.

76. Michael Löwe. Implementing algebraic specifications by graph transformation systems. *Journal on Information Processing and Cybernetics (EIK)*, 26(11/12):615–641, 1990.
77. Luc Maranget. Optimal derivations in weak lambda calculi and in orthogonal term rewriting systems. In *Proc. Annual ACM Symposium on Principles of Programming Languages*, pages 255–269. ACM Press, 1991.
78. Aart Middeldorp and Erik Hamoen. Counterexamples to completeness results for basic narrowing (extended abstract). In *Proc. Algebraic and Logic Programming*, pages 244–258. Springer Lecture Notes in Computer Science 632, 1992.
79. Aart Middeldorp and Erik Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.
80. Robin Milner. *Communication and Concurrency.* Prentice Hall International, 1989.
81. M.H.A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.
82. Enno Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136:333–360, 1994.
83. Enno Ohlebusch. Conditional term graph rewriting. In *Proc. Algebraic and Logic Programming*, volume 1298 of *Lecture Notes in Computer Science*, pages 144–158. Springer-Verlag, 1997.
84. Enno Ohlebusch. Modularity of termination for disjoint term graph rewrite systems: A simple proof. *EATCS Bulletin*, 66:171–177, 1998.
85. Peter Padawitz. Graph grammars and operational semantics. *Theoretical Computer Science*, 19:117–141, 1982.
86. David A. Plaisted. Equational reasoning and term rewriting systems. In Dov M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming,* volume 1, pages 273–364. Clarendon Press, 1993.
87. Rinus Plasmeijer and Marko van Eekelen. *Functional Programming and Parallel Graph Rewriting.* Addison-Wesley, 1993.
88. Detlef Plump. Graph-reducible term rewriting systems. In *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 622–636. Springer-Verlag, 1991.
89. Detlef Plump. Implementing term rewriting by graph reduction: Termination of combined systems. In *Proc. Conditional and Typed Rewriting Systems*, volume 516 of *Lecture Notes in Computer Science*, pages 307–317. Springer-Verlag, 1991.

90. Detlef Plump. Collapsed tree rewriting: Completeness, confluence, and modularity. In *Proc. Conditional Term Rewriting Systems*, volume 656 of *Lecture Notes in Computer Science*, pages 97–112. Springer-Verlag, 1993.
91. Detlef Plump. Evaluation of functional expressions by hypergraph rewriting. Dissertation, Universität Bremen, Fachbereich Mathematik und Informatik, 1993.
92. Detlef Plump. Critical pairs in term graph rewriting. In *Proc. Mathematical Foundations of Computer Science 1994*, volume 841 of *Lecture Notes in Computer Science*, pages 556–566. Springer-Verlag, 1994.
93. Detlef Plump. Simplification orders for term graph rewriting. In *Proc. Mathematical Foundations of Computer Science 1997*, volume 1295 of *Lecture Notes in Computer Science*, pages 458–467. Springer-Verlag, 1997.
94. Jean-Claude Raoult. On graph rewritings. *Theoretical Computer Science*, 32:1–24, 1984.
95. Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973.
96. Ronan Sleep, Rinus Plasmeijer, and Marko van Eekelen, editors. *Term Graph Rewriting: Theory and Practice*. John Wiley, 1993.
97. John Staples. Computation on graph-like expressions. *Theoretical Computer Science*, 10:171–185, 1980.
98. John Staples. Optimal evaluations of graph-like expressions. *Theoretical Computer Science*, 10:297–316, 1980.
99. John Staples. Speeding up subtree replacement systems. *Theoretical Computer Science*, 11:39–47, 1980.
100. Yoshihito Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
101. Yoshihito Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.