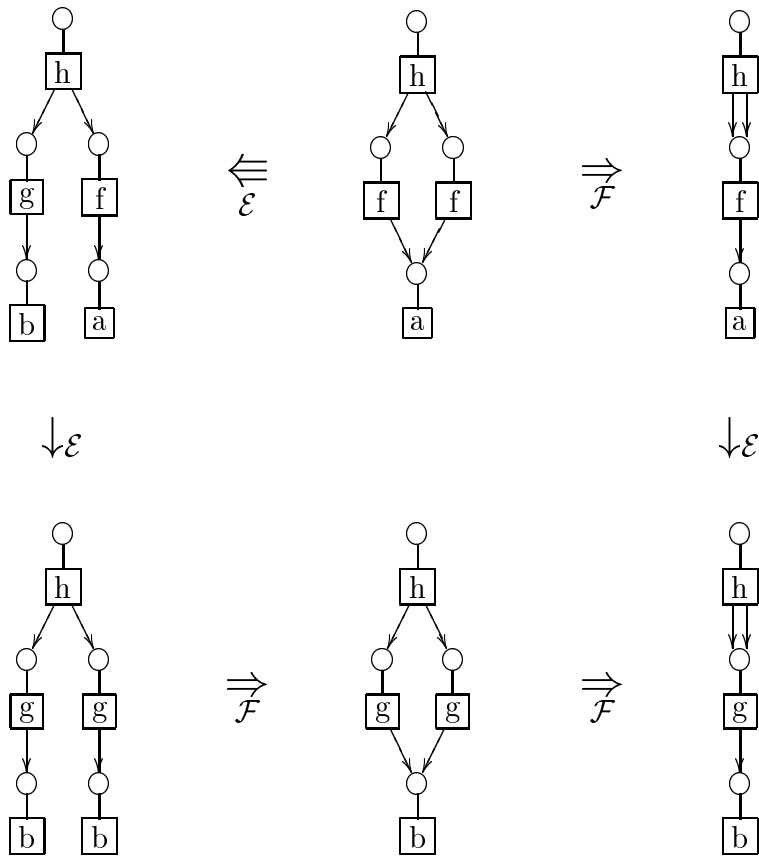


Evaluation of Functional Expressions by Hypergraph Rewriting



Detlef Plump

Evaluation of Functional Expressions by Hypergraph Rewriting

Vom Fachbereich Mathematik und Informatik
der Universität Bremen
zur Verleihung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)
genehmigte Dissertation

von

Detlef Plump

Bremen 1993

Gutachter:

Prof. Dr. Hans-Jörg Kreowski
Prof. Dr. Bernd Krieg-Brückner

Tag des Kolloquiums:

6. April 1993

Acknowledgements

I wish to thank my supervisor Hans-Jörg Kreowski and my co-supervisor Bernd Krieg-Brückner for giving me the opportunity to do research and for the freedom to pursue research topics of my own choice. Their working groups provided a stimulating and creative atmosphere.

Special thanks are due to Hans-Jörg, Annegret Habel, and Berthold Hoffmann for fruitful cooperation on jungle evaluation.

I am grateful to Annegret, Berthold, Frank Drewes, and Sabine Kuske for reading a draft of this thesis and making valuable comments.

I also like to thank my former colleagues of the PROSPECTRA project in Bremen for many discussions. In particular, I thank Berthold and Zhenyu Qian, my long-standing room-mate, for years of friendly disputes and discussions.

Contents

1	Introduction	1
1.1	Evaluation of Functional Expressions	1
1.2	Graphs vs. Trees	3
1.3	Contents of the Thesis	4
1.4	Related Work	5
2	Abstract Reduction Systems	8
3	Hypergraph Rewriting	11
3.1	Hypergraphs and Hypergraph Morphisms	12
3.2	Hypergraph Pushouts	13
3.3	Hypergraph Rules and Derivations	17
4	Representing Terms by Jungles	20
4.1	Jungles and Jungle Morphisms	20
4.2	Folding	26
5	Jungle Evaluation	30
5.1	Equations and Term Rewrite Rules	30
5.2	Evaluation Rules	32
5.3	Soundness	36
5.4	Applicability and Normal Forms	41
5.5	Termination	45
5.6	Confluence	49
6	Collapsed Tree Rewriting	53
6.1	Restriction and Extension of Derivations	54
6.2	Getting Rid of Garbage	57
6.3	Completeness	60
6.4	Termination	65
6.5	Confluence and Unique Normal Forms	68

7	Critical Pairs	72
7.1	Characterizing Local Confluence	72
7.2	The Critical Pair Lemma	77
8	Modularity	86
8.1	Modularity of Termination	88
8.2	Modularity of Convergence	92
9	Conclusion	97
9.1	Summary	97
9.2	Outlook	99
A	Basic Mathematical Notions and Notation	101
B	Proofs for Section 3.2	102
	Bibliography	105
	Index	115

Chapter 1

Introduction

To execute programs and specifications in declarative programming and specification languages means to evaluate functional and logical expressions by transformation rules such as equations or clauses. The efficient realization of the latter is the central problem in the implementation of declarative languages. The present thesis is concerned with the evaluation of functional expressions as they occur in functional and logic programming, algebraic specification, theorem proving, and symbolic computation. An approach is proposed to represent expressions by (hyper-)graphs and to evaluate these by transformation rules derived from directed equations. The emphasis is laid on the development of a corresponding theory which provides precise results about soundness, completeness, and operational properties like termination and confluence.

This introduction reviews some appearances of expression evaluation in computer science, motivates the use of graph rewriting, gives an overview over the contents of the thesis, and outlines related work.

1.1 Evaluation of Functional Expressions

Expressions consisting of functions, constants, and variables occur in virtually all higher programming languages. In PASCAL, for example, there are arithmetic expressions like

$$(x + 1) \times (y \bmod 2)$$

which may contain predefined functions (here $+$, \times , and \bmod) as well as functions programmed by the user. The evaluation costs at run time for such expressions crucially depend on the quality of the assembly or machine code generated by the compiler.

For functional programming languages, such as ML and MIRANDA,¹ func-

¹Information on ML and MIRANDA can be found, for example, in the textbooks of Paulson [Pau91] and Holyer [Hol91].

tional expressions are a central concept in that they constitute input and output of program executions. Functional programs consist of function definitions in form of equations which may contain recursive function calls. The Fibonacci function, for instance, can be defined by the following three equations:

$$\begin{aligned}\text{fib}(0) &= 0 \\ \text{fib}(1) &= 1 \\ \text{fib}(x + 2) &= \text{fib}(x) + \text{fib}(x + 1)\end{aligned}$$

Execution of functional programs consists in evaluation of expressions by applying function definitions until an expression is obtained that cannot be further transformed and which is the result of the computation. As an example, the expression $\text{fib}(3)$ may be evaluated by the steps shown in Figure 1.1.

$$\begin{aligned}\text{fib}(3) &\rightarrow \text{fib}(1) + \text{fib}(2) \\ &\rightarrow 1 + \text{fib}(2) \\ &\rightarrow 1 + \text{fib}(0) + \text{fib}(1) \\ &\rightarrow 1 + 0 + \text{fib}(1) \\ &\rightarrow 1 + \text{fib}(1) \\ &\rightarrow 1 + 1 \\ &\rightarrow 2\end{aligned}$$

Figure 1.1: Evaluation of $\text{fib}(3)$

An essential difference between procedural languages like PASCAL and (pure) functional languages is that expression evaluation in the former may produce side effects by altering the state of variables while in the latter there is no state. The absence of state eases the formal reasoning about programs since expressions can be transformed by algebraic laws without to change their meaning. (See [Hug89] for a discussion on the virtues of functional languages.)

A further field of computer science in which functional expressions play a fundamental role is algebraic specification (see [EM85, Wir90] for introductions). Similar to functional programs, equations are the key concept of algebraic specifications (in their basic form), with the difference that arbitrary expressions may occur not only on the right-hand sides but also on the left-hand sides of equations. Such specifications are primarily intended to specify abstract data types by fixing classes of models, but can also be used operationally for at least two purposes. Firstly, specifications may serve as (in general nondeterministic) functional programs which are executed as described above (see for example [FGKM85, O'D85, GH86]). Secondly, logical consequences of a given specification in form of equations may, under certain conditions, be proved by evaluating both sides of equations to the same result (see [KB70, Hue80, Bac91]).

The theoretical basis for expression evaluation in (first-order) functional programming and algebraic specification is provided by the theory of *term rewriting systems*. These are sets of directed equations which are applied to expressions in only one direction (see Section 5.1 and the references given there). In this thesis, term rewriting systems are the point of departure for expression evaluation by hypergraph rewriting: term rewrite rules are translated into hypergraph rewrite rules, and the computational model of term rewriting serves as a “reference model” for comparing properties and results.

1.2 Graphs vs. Trees

A serious drawback of term rewriting is that its (one-to-one) implementation requires the copying of subexpressions, causing sometimes a multiplication of evaluation work in a single step. Consider, for instance, the definition of a function “double” by

$$\text{double}(x) = x + x$$

and an expression $\text{double}(E)$ with E standing for an arbitrary subexpression. Applying the definition yields $E + E$, so that E has to be evaluated twice if expressions are represented as strings or trees (see Figure 1.2). In contrast, duplication of work is avoided by using a graph representation with two pointers to a single occurrence of E (Figure 1.3). This simple example already reveals the

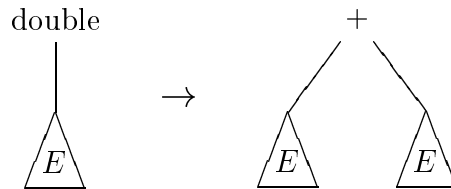


Figure 1.2: A tree rewrite step

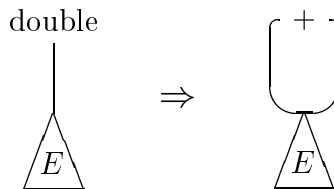


Figure 1.3: A graph rewrite step

essential advantage of graph rewriting: substructures need not be copied but can be shared by introducing additional pointers.

In the present thesis, a formal approach to the evaluation of functional expressions by (hyper-)graph rewriting is developed. The formal treatment allows to prove results about soundness, completeness, termination, confluence, etc. and to compare properties in a precise way with the corresponding properties of the conventional term rewriting approach.

The approach presented here grew out of work reported in the papers [HKP88, HP88a, HP88b, HKP91, HP91, Plu91a, Plu91b, Plu93a, Plu93b], in which parts of the results of this thesis are published.

1.3 Contents of the Thesis

Chapter 2 briefly reviews notions and facts for abstract reduction systems. These provide a general setting for studying properties common to rewriting systems over arbitrary domains and thereby allow to separate general properties of binary relations from those inherent in particular models such as term or hypergraph rewriting.

Chapter 3 introduces the “Berlin approach” to graph rewriting, adapted to the hypergraph case. The computational models “jungle evaluation” and “collapsed tree rewriting” developed in Chapters 5 and 6 are both based on the Berlin approach.

In Chapter 4, jungles are defined as special hypergraphs for representing (sets of) terms, and the relationship between jungles and terms is worked out. Moreover, the concept of folding is introduced which allows to transform jungles such that the degree of sharing is enhanced.

The realization of term evaluation by hypergraph rewriting is developed in Chapter 5. It is shown that the resulting model of *jungle evaluation* is sound with respect to term rewriting. Furthermore, the applicability of jungle evaluation rules is analyzed in order to clarify the relation between jungle and term normal forms. Next termination is considered and it turns out that jungle evaluation is terminating whenever term rewriting is, but not vice versa. To guarantee that confluence carries over from term rewriting to jungle evaluation, the latter must be normalizing and one has to ignore garbage consisting of certain unreachable nodes and edges.

Because of problems with confluence and completeness, jungle evaluation is modified in Chapter 6 by building in garbage collection. This is accomplished by removing garbage after every evaluation step. The resulting model of *collapsed tree rewriting* is shown to be complete with respect to equational validity in the same sense as term rewriting is. With respect to termination, collapsed tree rewriting and jungle evaluation are equivalent. The class of rule systems that are

confluent under collapsed tree rewriting turns out to be a proper subclass of those systems for which term rewriting is confluent. Nevertheless, collapsed tree rewriting has unique normal forms if and only if term rewriting has. As a consequence, confluence of term rewriting carries over to collapsed tree rewriting whenever the latter is normalizing. Convergent—that is confluent and terminating—term rewriting systems are shown to form a proper subclass of those systems that are convergent under collapsed tree rewriting.

In Chapter 7, a criterion for local confluence is developed by considering “critical overlaps” of left-hand sides of evaluation rules. One has to require that the jungles produced by *critical pairs* (overlapping evaluation steps) can be rewritten to descendants that coincide up to certain garbage. The sufficiency of this condition is stated in the so-called Critical Pair Lemma. As a result, terminating systems enjoy a characterization of confluence through critical pairs and a decision procedure for confluence.

Two modularity results for collapsed tree rewriting are established in Chapter 8. It turns out that the combination of rule sets with disjoint function symbols preserves termination, in contrast to the situation for term rewriting. In fact, this still holds when the involved systems have certain function symbols in common. The second result reveals that collapsed tree rewriting behaves modular with respect to convergence, which again does not hold for term rewriting. These properties give evidence that expression evaluation by collapsed tree rewriting has advantages over term rewriting not only for efficiency reasons but also with respect to the incremental construction of terminating and convergent systems.

Chapter 9 gives a résumé of this thesis and outlines some topics for future research.

1.4 Related Work

Wadsworth [Wad71] was the first who described expression evaluation by graph rewriting. He considered the implementation of β -reduction in the lambda calculus. This problem was also addressed by Staples [Sta79, Sta83], Kathail [Kat90], Lamping [Lam90], and Gonthier, Abadi and Lévy [GAL92], who aimed at implementing optimal (i.e. shortest) reduction sequences.

Closer to the subject of this thesis, Staples [Sta80a, Sta80b, Sta80c] developed a formalism for the manipulation of “expression graphs” which represent ordinary first-order terms. He could give an optimal evaluation strategy for a certain class of term rewriting systems with non-overlapping left-hand sides.

Barendregt, van Eekelen, Glauert, Kennaway, Plasmeijer, and Sleep [BvEG⁺87] presented a translation of term rewriting systems into graph rewriting systems, leading to a computational model called *term graph rewriting*. They showed the soundness of this model and that full substitution is a normalizing rewrite strat-

egy for the case of left-linear, non-overlapping term rewriting systems. (Several papers on term graph rewriting and related approaches can be found in [SPvE93].)

Yet another graph rewrite formalism was devised by Farmer and Watro [FW90] to allow a very liberal kind of rewrite steps that are sound with respect to term rewriting (see also page 34).

The evaluation of cyclic term graphs is investigated by Kennaway, Klop, Sleep, and de Vries [KKSdV93]. Their main result says that finitary cyclic graph rewriting is “adequate” for rational transfinite term rewriting.

Goguen, Kirchner, Meseguer, and Viry [GKM87, KV90, Vir92] proposed to use graph rewriting in the implementation of term rewriting on parallel machine architectures. They considered various practical and theoretical problems for realizing non-sequential computations on term graphs.

The approaches mentioned so far introduce notions of graph rewriting that are tailored to expression evaluation. Another line of research, beginning with Ehrig and Rosen [ER76], models expression evaluation within the Berlin approach to general graph rewriting. In [ER76] it was shown that expression evaluation by graph rewriting yields unique results for a restricted class of function definitions. Padawitz [Pad82] provided expression graphs with a functional semantics based on an interpretation of variable nodes in certain algebras. He presented a class of graph rewrite rules which are correct in the sense that they preserve the semantics, and exploited this result to prove the correctness of a graph based LISP interpreter. Parisi-Presicce, Ehrig, and Montanari [PPEM87] extended the Berlin approach by a concept of graph variables and simulated term rewrite steps within this setting. Löwe [Löw90] used slightly modified jungles together with general graph rewrite rules to construct algebras that implement algebraic specifications. For linear equations he gave a standard construction of a graph algebra that is initial among all models of a specification.

Some other papers along the lines of the Berlin approach also used jungles to represent terms or formulas. Corradini, Ehrig, Löwe, Montanari, Parisi-Presicce, and Rossi [CMR⁺91, CRPP91] modelled the execution of logic programs by jungle rewriting, using a translation of logical clauses into jungle rewrite rules. Their central idea is to define rewrite steps directly in the category of jungles instead of using the conventional category of hypergraphs. This allows to exploit a correspondence between the pushout construction for jungles and the unification of terms. Corradini and Rossi [CR93a, CR93b] extended this approach such that term rewriting systems can also be implemented.

Hoffmann [Hof92] enriched the jungle evaluation approach of [HP91] by “memoization”—an optimization technique developed for functional programming languages—in order to avoid the reevaluation of subexpressions in different evaluation steps. He showed the soundness of this extension for non-overlapping, non-looping term rewriting systems.

Finally, it should be mentioned that graph rewriting (in this context usu-

ally called graph reduction) has become a common implementation technique for (higher-order) functional programming languages. The basic idea, going back to Turner [Tur79], is to transform expressions of the lambda calculus into so-called combinator terms which are evaluated by graph rewriting. See [Pey87] for a textbook on this technique and the proceedings [FK87] for various practical aspects of graph reduction.

Chapter 2

Abstract Reduction Systems

Rewriting systems (also called reduction or replacement systems) are means to compute by stepwise transformation of objects. These objects may be strings, terms, formulas, graphs or any other entities from a given domain. The present thesis is concerned with rewriting systems over three different domains, consisting of terms, “jungles”, and “collapsed trees”. Several concepts and properties of rewriting systems can be defined and studied independently from specific domains, in the setting of so-called abstract reduction systems. These are just sets with a binary relation which represents elementary transformation steps. Abstract reduction systems were studied the first time by Newman [New42] and systematically applied in influential papers of Rosen [Ros73] and Huet [Hue80] to separate abstract properties of relations from properties depending on the structure of objects.

In the following some basic notions and facts for abstract reduction systems are collected. Further concepts and results can be found, for example, in [BO93, DJ90, Hue80, Jan88, Klo92]. The terminology used here is consistent with Dershowitz’s and Jouannaud’s survey [DJ90].

An *abstract reduction system* is a set A together with a binary relation \rightarrow on A . For this chapter, let $\langle A, \rightarrow \rangle$ be an arbitrary abstract reduction system. Given elements a, b in A such that $\langle a, b \rangle \in \rightarrow$, this is denoted by $a \rightarrow b$.

Various closure relations over \rightarrow are written as follows:

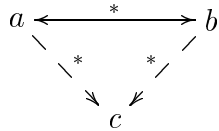
- $\xrightarrow{\lambda}$ reflexive closure
- \leftrightarrow symmetric closure
- $\xrightarrow{\pm}$ transitive closure
- $\xrightarrow{*}$ transitive-reflexive closure
- $\xleftrightarrow{*}$ symmetric-transitive-reflexive closure

The inverse relation of \rightarrow is denoted by \leftarrow . Two elements a, b in A with $a \leftrightarrow^* b$ are *convertible*. Elements a, b have a *common reduct* if $a \rightarrow^* c \leftarrow^* b$ for

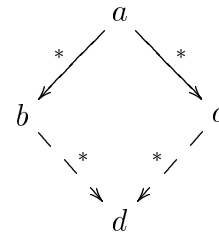
some c . An element a is a *normal form* if there is no b such that $a \rightarrow b$. Element a has a *normal form* if $a \rightarrow^* b$ for some normal form b .

The relation \rightarrow is

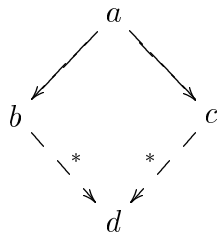
- (1) *terminating* if there is no infinite sequence $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \dots$,
- (2) *normalizing* if each element in A has a normal form,
- (3) *Church-Rosser* if for all a, b with $a \leftrightarrow^* b$ there is some c such that $a \rightarrow^* c \leftarrow^* b$ (see Figure 2.1(a)),
- (4) *confluent* if for all a, b, c with $b \leftarrow^* a \rightarrow^* c$ there is some d such that $b \rightarrow^* d \leftarrow^* c$ (see Figure 2.1(b)),
- (5) *locally confluent* if for all a, b, c with $b \leftarrow a \rightarrow c$ there is some d such that $b \rightarrow^* d \leftarrow^* c$ (see Figure 2.1(c)),
- (6) *strongly confluent* if for all a, b, c with $b \leftarrow a \rightarrow c$ there is some d such that $b \rightarrow^\lambda d \leftarrow^\lambda c$ (see Figure 2.1(d)),
- (7) *convergent* if it is terminating and confluent.



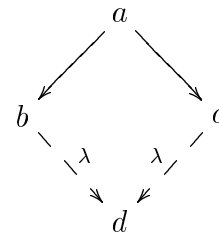
(a) Church-Rosser



(b) confluent



(c) locally confluent



(d) strongly confluent

Figure 2.1: Confluence properties

The following lemma lists some relations between properties of abstract reduction systems. From statement (5) the main benefit of confluent systems can be concluded: two elements having normal forms are convertible if and only if their normal forms are identical (so, in particular, every element has at most one normal form).

Lemma 2.1 (1) *Termination implies normalization.*

(2) *The Church-Rosser property is equivalent to confluence.*

(3) *Strong confluence 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 $a = b$.*

The implications (1) and (4) are obvious. Concerning (2), the Church-Rosser property clearly implies confluence while the converse is shown by induction on the number of \leftrightarrow -steps constituting an equivalence $a \leftrightarrow^* b$. Statement (3) is proved by two inductions, the first showing that if \rightarrow is strongly confluent, then for all a, b, c with $b \leftarrow a \rightarrow^* c$ there is some d such that $b \rightarrow^* d \leftarrow^* c$, while the second induction shows that the latter property implies confluence. Finally, proposition (5) is a simple consequence of (2).

The converses of (1), (3), (4), and (5) do not hold. A well-known counterexample for the converse of (4) is the relation shown in Figure 2.2. However, the situation changes in the presence of termination, as the following important result shows.

Lemma 2.2 (Newman's Lemma [New42])

A terminating relation is confluent if and only if it is locally confluent.

An elegant and simple proof of this lemma is given by Huet [Hue80]. Note that termination cannot be relaxed to normalization, as is demonstrated by the normalizing relation in Figure 2.2.

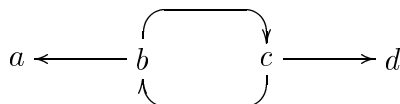


Figure 2.2: Local confluence without confluence

Chapter 3

Hypergraph Rewriting

The computational models “jungle evaluation” and “collapsed tree rewriting” introduced in subsequent chapters are both based on the Berlin approach to graph rewriting¹. This chapter reviews fundamentals of the Berlin approach and, at the same time, generalizes it to hypergraph rewriting. Hypergraphs are more appropriate for the evaluation of functional expressions since they allow a natural representation of expressions. This is because an n -place function symbol can be realized as a single hyperedge with one source node and n target nodes, while in the graph case a node (or an edge) together with n additional edges has to be used.

The lifting of graph rewriting to hypergraph rewriting is conceptually simple since the essential concepts in the Berlin approach have abstract descriptions by categorical notions (beside concrete set-theoretical descriptions). One just replaces the underlying category of graphs and graph morphisms by a category of hypergraphs and hypergraph morphisms. This is in the spirit of [EHKPP91], where the Berlin approach is generalized to “high-level replacement systems” over arbitrary categories.²

The only difficulty in passing from graphs to hypergraphs is to redefine and prove correct the construction of pushouts and pushout complements. These proofs are simple but tedious; they are deferred to Appendix B.

For a comprehensive introduction to the Berlin approach to graph rewriting, which includes motivations and references to the original literature, the reader is referred to Ehrig’s survey [Ehr79]. Recent developments can be found in Habel’s survey [Hab92] and in the workshop proceedings [ENR83, ENRR87, EKR91].

¹More precisely, the so-called “double-pushout approach” is used. As an equally suited alternative, the “single-pushout approach” recently developed by Löwe and Ehrig [LE91, Löw93] could have been chosen.

²The hypergraph category HYP-GRAPHS and the associated class of rule morphisms used in [EHKPP91] are more restrictive than their counterparts here, as hypergraphs are untyped and morphisms in rules have to be injective.

3.1 Hypergraphs and Hypergraph Morphisms

A *signature* $\Sigma = \langle \Sigma_V, \Sigma_E, Type_\Sigma \rangle$ consists of two sets Σ_V, Σ_E of *labels* and a mapping $Type_\Sigma: \Sigma_E \rightarrow \mathcal{P}(\Sigma_V^* \times \Sigma_V^*)$. For $\sigma \in \Sigma_E$, $Type_\Sigma(\sigma)$ specifies the admissible pairs of label strings for source and target nodes of hyperedges labelled with σ . Let Σ be fixed for the rest of this chapter.

A *hypergraph* over Σ is a system $G = \langle V, E, l, m, s, t \rangle$ consisting of

- two finite sets V and E of *nodes* (or *vertices*) and *hyperedges* (or *edges*),
- two mappings $l: V \rightarrow \Sigma_V$ and $m: E \rightarrow \Sigma_E$ providing nodes and hyperedges with labels, and
- two mappings $s, t: E \rightarrow V^*$ which assign strings $s(e)$ and $t(e)$ of *source nodes* and *target nodes* to each hyperedge e such that $\langle l^*(s(e)), l^*(t(e)) \rangle \in Type_\Sigma(m(e))$.

The above type concept generalizes the hypergraph definition of Habel [Hab89] in that it allows to “customize” the class of hypergraphs (beside the extension that nodes are labelled). Untyped hypergraphs can be obtained as a special case by letting $Type_\Sigma(\sigma) = \Sigma_V^* \times \Sigma_V^*$ for each $\sigma \in \Sigma_E$. Courcelle [Cou90] imposes a type structure on hypergraphs (without node labels) by requiring that the label of a hyperedge uniquely determines the number of incident nodes. The present definition is more general as it allows to “overload” hyperedge labels by assigning types with more than one element (although types are always singletons in the following chapters, see Section 4.1).

In pictures of hypergraphs, nodes are drawn as circles or ovals, and hyperedges as boxes, in both cases with inscribed labels. Lines connect a hyperedge with its source nodes, while arrows point to the target nodes. Lines and arrows are numbered corresponding to the order of nodes in source and target strings. Figure 3.1 shows a hyperedge together with its source and target nodes, where $\langle a_1 \dots a_m, b_1 \dots b_n \rangle \in Type_\Sigma(\sigma)$.

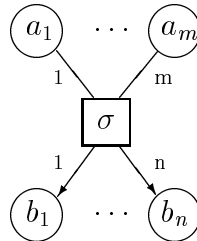


Figure 3.1: Hyperedge with source and target nodes

The components of a hypergraph G are referred to as V_G , E_G , l_G , m_G , s_G , and t_G . A hyperedge e in G is *incident* to each node in $s_G(e)$ and $t_G(e)$. For a node v in G , $\text{indegree}_G(v) = \sum_{e \in E_G} \text{occ}(v, t_G(e))$, where $\text{occ}(v, t_G(e))$ is the number of occurrences of v in $t_G(e)$. $\text{outdegree}_G(v)$ is defined analogously.

Let $v, v' \in V_G$. v is a *predecessor* of v' if there is $e \in E_G$ such that $v \in s_G(e)$ and $v' \in t_G(e)$. The relations $>_G$ and \geq_G are the transitive respectively transitive-reflexive closure of the predecessor relation. If $v \geq_G v'$, then v' is *reachable* from v . G is *acyclic* if there is no $v \in V_G$ with $v >_G v$.

Let H be a hypergraph with $V_G \subseteq V_H$, $E_G \subseteq E_H$. Then G is a *subhypergraph* of H , denoted by $G \subseteq H$, if l_G , m_G , s_G , and t_G are restrictions of the corresponding mappings of H . Given a subset V of V_H , the subhypergraph U of H with $V_U = V$ and $E_U = \{e \in E_H \mid s_H(e), t_H(e) \in V^*\}$ is said to be *induced* by V .

A *hypergraph morphism* f from a hypergraph A to a hypergraph B is a pair of functions $\langle f_V: V_A \rightarrow V_B, f_E: E_A \rightarrow E_B \rangle$, denoted by $f: A \rightarrow B$, such that

$$\begin{aligned} l_B \circ f_V &= l_A, m_B \circ f_E = m_A && \text{(preservation of labels) and} \\ s_B \circ f_E &= f_V^* \circ s_A, t_B \circ f_E = f_V^* \circ t_A && \text{(preservation of source and target nodes).} \end{aligned}$$

A and B are called the *domain* respectively *codomain* of f . f is *injective* (*surjective*) if f_V and f_E are injective (surjective). f is an *isomorphism* if it is injective and surjective; in this case A and B are *isomorphic*, denoted by $A \cong B$.

For a subhypergraph U of A , fU denotes the subhypergraph of B with node set $f_V V_U$ and edge set $f_E E_U$. Given a subhypergraph W of B with $fU \subseteq W$, the *restriction* $U \rightarrow W$ of f and the *inclusions* $U \rightarrow A$, $W \rightarrow B$ have the obvious definitions. The *composition* $g \circ f: A \rightarrow C$ of two hypergraph morphisms $f: A \rightarrow B$ and $g: B \rightarrow C$ is defined by $g \circ f = \langle g_V \circ f_V, g_E \circ f_E \rangle$.

When nodes and edges of hypergraphs can be treated analogously, a notation without references to node and edge sets is used. For example, “for all $x, y \in G$, $f(x) = f(y)$ implies $x = y$ ” is an abbreviation for “for all $x, y \in V_G$, $f_V(x) = f_V(y)$ implies $x = y$, and for all $x, y \in E_G$, $f_E(x) = f_E(y)$ implies $x = y$ ”.

3.2 Hypergraph Pushouts

The basic idea of the Berlin approach is to describe the transformation of graphs by *pushouts*. The concept of a pushout, which originates from category theory, allows to describe the deletion, identification, and insertion of nodes and edges on an abstract level. In this section set-theoretical constructions of pushouts and pushout complements for hypergraphs are given. Together they constitute the construction of a hypergraph rewrite step (given in the next section).

Let $b: A \rightarrow B$, $c: A \rightarrow C$ be two hypergraph morphisms with a common domain. Then a hypergraph D together with two hypergraph morphisms $f: C \rightarrow D$, $g: B \rightarrow D$ is a *pushout* of b and c if the following two conditions are satisfied:

Commutativity. $g \circ b = f \circ c$.

Universal property. For all hypergraphs D' and hypergraph morphisms $f': C \rightarrow D'$, $g': B \rightarrow D'$ with $g' \circ b = f' \circ c$ there exists a unique hypergraph morphism $h: D \rightarrow D'$ such that $h \circ f = f'$ and $h \circ g = g'$.

The diagram in Figure 3.2 is also called a pushout when the above conditions are satisfied. It is shown below that D can be constructed from B by (1) identifying

$$\begin{array}{ccc} A & \xrightarrow{c} & C \\ b \downarrow & & \downarrow f \\ B & \xrightarrow{g} & D \end{array}$$

Figure 3.2: Pushout diagram

all items the origins of which in A are identified by c , and (2) adding all nodes and edges from $C - cA$. So, intuitively, D is the “gluing” of B and C in the common interface A .

By replacing the notions “hypergraph” and “hypergraph morphism” in the above definition by “set” respectively “function”, one obtains the definition of set pushouts. In the construction of hypergraph pushouts, set pushouts are constructed separately for nodes and edges.

The standard construction for the pushout of two functions $b: A \rightarrow B$ and $c: A \rightarrow C$ is to build the disjoint union $B + C$ and then to form the quotient $(B + C)/\equiv$, where \equiv is the equivalence generated by $\{(b(a), c(a)) \mid a \in A\}$ (see for example [RB88]). The following construction is more efficient because it avoids the copying of the subset cA and its subsequent gluing with bA .

Construction 3.1 (set pushout) Let $b: A \rightarrow B$ and $c: A \rightarrow C$ be functions. Define the relation \sim on B by

$$x \sim y \text{ if } x = y \text{ or there are } \bar{x}, \bar{y} \in A \text{ with } b(\bar{x}) = x, b(\bar{y}) = y, \text{ and } c(\bar{x}) = c(\bar{y}).$$

Let \approx be the transitive closure of \sim . The relation \approx is an equivalence relation because \sim is reflexive and symmetric. Let B/\approx be the quotient of B and $[]: B \rightarrow B/\approx$ be the associated natural function which sends each element to its equivalence class. Define the set D by

$$D = B/\approx + (C - cA)$$

and the functions $g: B \rightarrow D$ and $f: C \rightarrow D$ by

$$\begin{aligned} g(x) &= [x], \\ f(x) &= \begin{cases} x & \text{if } x \in C - cA, \\ [b(\bar{x})] & \text{if } x = c(\bar{x}) \text{ for some } \bar{x} \in A. \end{cases} \end{aligned}$$

Lemma 3.2 *With the notations of Construction 3.1, the diagram in Figure 3.2 is a set pushout.*

Proof See Appendix B. □

Construction 3.3 (hypergraph pushout) Let $b: A \rightarrow B$ and $c: A \rightarrow C$ be hypergraph morphisms. Define $D = \langle V_D, E_D, l_D, m_D, s_D, t_D \rangle$ as follows. $V_D = V_B / \approx_V + (V_C - c_V V_A)$ together with $f_V: V_C \rightarrow V_D$ and $g_V: V_B \rightarrow V_D$ is the set pushout of b_V and c_V as constructed in 3.1, and $E_D = E_B / \approx_E + (E_C - c_E E_A)$ together with $f_E: E_C \rightarrow E_D$ and $g_E: E_B \rightarrow E_D$ is the set pushout of b_E and c_E . The functions $l_D: V_D \rightarrow \Sigma_V$, $m_D: E_D \rightarrow \Sigma_E$, and $s_D, t_D: E_D \rightarrow V_D^*$ are defined by

$$\begin{aligned} l_D(v) &= \begin{cases} l_C(v) & \text{if } v \in V_C - c_V V_A, \\ l_B(\bar{v}) & \text{if } v = [\bar{v}] \in V_B / \approx_V, \end{cases} \\ m_D(e) &= \begin{cases} m_C(e) & \text{if } e \in E_C - c_E E_A, \\ m_B(\bar{e}) & \text{if } e = [\bar{e}] \in E_B / \approx_E, \end{cases} \\ s_D(e) &= \begin{cases} f_V^*(s_C(e)) & \text{if } e \in E_C - c_E E_A, \\ g_V^*(s_B(\bar{e})) & \text{if } e = [\bar{e}] \in E_B / \approx_E, \end{cases} \end{aligned}$$

and analogously for t_D . Moreover, let $f = \langle f_V, f_E \rangle$ and $g = \langle g_V, g_E \rangle$.

Lemma 3.4 *With the notations of Construction 3.3, the diagram in Figure 3.2 is a hypergraph pushout.*

Proof See Appendix B. □

The above constructions show that both the identification of items in a hypergraph B and the addition of items to B can be described by a pushout of two appropriate hypergraph morphisms. The resulting hypergraph is unique up to isomorphism, as can be easily shown by the universal property of pushouts.

The deletion of parts of a hypergraph can also be described by a pushout. Consider two hypergraph morphisms $c: A \rightarrow C$ and $f: C \rightarrow D$. Then a hypergraph B together with two hypergraph morphisms $b: A \rightarrow B$, $g: B \rightarrow D$ is a *pushout complement* of c and f , if D together with f and g is a pushout of b and c . B can be obtained from D by deleting certain nodes and edges, provided there exists a pushout complement at all.

For simplicity, the construction of pushout complements in the next lemma requires that one of the given hypergraph morphisms is an inclusion. This suffices for the kind of hypergraph rewrite steps that are used in the following chapters.

Let $g: L \rightarrow G$ be a hypergraph morphism and K be a subhypergraph of L . Then the *gluing condition* consists of the following two conditions:

Contact condition. No edge in $E_G - E_{gL}$ is incident to any node in $V_{gL} - V_{gK}$.

Identification condition. For all $x, y \in L$, $g(x) = g(y)$ implies $x = y$ or $x, y \in K$.

Lemma 3.5 (pushout complement) *Let $g: L \rightarrow G$ be a hypergraph morphism and $K \rightarrow L$ be the inclusion of a subhypergraph K in L such that the gluing condition is satisfied. Then the diagram*

$$\begin{array}{ccc} L & \longleftarrow & K \\ g \downarrow & & \downarrow d \\ G & \longleftarrow & D \end{array}$$

is a hypergraph pushout, where $D \subseteq G$ has node and edge sets $G - (gL - gK)$, $D \rightarrow G$ is the associated inclusion, and d is the restriction of g .

Proof See Appendix B. □

It can be shown that the gluing condition is not only sufficient but also necessary for the existence of pushout complements. Moreover, the injectivity of $K \rightarrow L$ ensures that D is unique up to isomorphism. (See [Ros75b] for proofs in the graph case.)

The next two lemmas list some pushout properties that are used in the sequel.

Lemma 3.6 (pushout properties) *Let*

$$\begin{array}{ccc} A & \xrightarrow{c} & C \\ b \downarrow & & \downarrow f \\ B & \xrightarrow[g]{} & D \end{array}$$

be a hypergraph pushout. Then the following holds:

- (1) *For every isomorphism $i: D \rightarrow D'$, D' together with $i \circ f$ and $i \circ g$ is a pushout of b and c .*
- (2) *For every pushout $\langle D', f': C \rightarrow D', g': B \rightarrow D' \rangle$ of b and c , the unique hypergraph morphism $i: D \rightarrow D'$ satisfying $i \circ f = f'$ and $i \circ g = g'$ is an isomorphism.*

(3) $gB \cup fC = D$ and $gB \cap fC = gbA$, for nodes and edges separately.

(4) If b is injective (surjective), then so is f .

Proof Properties (1) and (2) are directly shown by the commutativity and universal properties of pushouts. Properties (3) and (4) are easily checked for the pushout construction 3.3 (using 3.1) and carry over to arbitrary hypergraph pushouts by (2). \square

Lemma 3.7 (composed pushouts) *Let*

$$\begin{array}{ccccc}
 A & \longrightarrow & B & \longrightarrow & C \\
 \downarrow & & \downarrow & & \downarrow \\
 & [1] & & [2] & \\
 D & \longrightarrow & F & \longrightarrow & G
 \end{array}$$

be a commutative diagram of hypergraph morphisms. Then the following holds:

(1) If the squares [1] and [2] are pushouts, then so is the outer rectangle $[1] \cup [2]$.

(2) If $[1] \cup [2]$ and [1] are pushouts, then so is [2].

(3) If $[1] \cup [2]$ and [2] are pushouts and $B \rightarrow C$ is injective, then [1] is a pushout.

Proof Properties (1) and (2) are standard facts in category theory (see for example [AHS90]) which are straightforward shown by the definition of pushouts. A proof of (3) in the graph case—which holds also for hypergraphs—can be found in [EK79]. \square

3.3 Hypergraph Rules and Derivations

A rule $p = (L \leftarrow K \rightarrow R)$ consists of two hypergraph morphisms with a common domain. L is the *left-hand side*, R the *right-hand side*, and K the *interface* of p .

Let G, H be hypergraphs and $p = (L \leftarrow K \rightarrow R)$ be a rule. G *directly derives* H *through* p , denoted by $G \Rightarrow_p H$, if there are two hypergraph pushouts of the form shown in Figure 3.3.

If $K \rightarrow L$ is injective, then G, p , and $L \rightarrow G$ determine D and H uniquely up to isomorphism. From now on it is assumed that $K \rightarrow L$ is an inclusion, for every rule $(L \leftarrow K \rightarrow R)$. Accordingly, D is always assumed to be the subhypergraph of G constructed in Lemma 3.5.

$$\begin{array}{ccccc}
L & \longleftarrow & K & \longrightarrow & R \\
\downarrow & & \downarrow & & \downarrow \\
G & \longleftarrow & D & \longrightarrow & H
\end{array}$$

Figure 3.3: Direct derivation

Using the constructions of the previous section, the application of a rule $p = (L \leftarrow K \rightarrow R)$ to a hypergraph G amounts to the following steps:

- (1) Find a hypergraph morphism $g: L \rightarrow G$.
- (2) Check the gluing condition.
- (3) Remove $gL - gK$ from G , yielding a hypergraph D , a hypergraph morphism $K \rightarrow D$ (which is the restriction of g), and the inclusion $D \rightarrow G$.
- (4) Construct the pushout of $K \rightarrow D$ and $K \rightarrow R$, yielding a hypergraph H and hypergraph morphisms $R \rightarrow H$, $D \rightarrow H$.

The notation $G \Rightarrow_{p,g} H$ is used when $g: L \rightarrow G$ shall be made explicit. Let \mathcal{S} be a set of hypergraph rules. Let \mathcal{S} be a set of hypergraph rules. Then $G \Rightarrow_{\mathcal{S}} H$ expresses that there is some rule p in \mathcal{S} such that $G \Rightarrow_p H$. $G \Rightarrow_{\mathcal{S}}^{\lambda} H$ means that $G \Rightarrow_{\mathcal{S}} H$ or $G \cong H$. G derives H , denoted by $G \Rightarrow_{\mathcal{S}}^* H$, if $G \cong H$ or there are hypergraphs G_0, \dots, G_n ($n \geq 1$) such that $G = G_0 \Rightarrow_{\mathcal{S}} G_1 \Rightarrow_{\mathcal{S}} \dots \Rightarrow_{\mathcal{S}} G_n = H$. The subscript \mathcal{S} is sometimes omitted, provided the set of rules is irrelevant or clear from the context. The inverses of the above relations are denoted by reversing the respective double arrow.

By Lemma 3.6(1), the above rewrite relations are “closed under isomorphism”, that is, for arbitrary hypergraphs G, G', H, H' it holds true that

$$G' \cong G \Rightarrow H \cong H' \text{ implies } G' \Rightarrow H',$$

and analogously for the relations \Rightarrow^{λ} and \Rightarrow^* . Thus \Rightarrow , \Rightarrow^{λ} , and \Rightarrow^* can be considered as relations on isomorphism classes of hypergraphs. This is the general view taken in this thesis, meaning that hypergraph rewriting over a given rule set \mathcal{S} is identified with the abstract reduction system $\langle \mathcal{H}, \Rightarrow_{\mathcal{S}} \rangle$ in which \mathcal{H} is the set of isomorphism classes of hypergraphs over Σ . Note that in this way $\Rightarrow_{\mathcal{S}}^{\lambda}$ and $\Rightarrow_{\mathcal{S}}^*$ become the reflexive respectively transitive-reflexive closure of $\Rightarrow_{\mathcal{S}}$, so that the considerations of the previous chapter apply. Nevertheless, in the following all definitions, constructions, and results are formulated for “concrete” hypergraphs which should be considered as representatives of their isomorphism classes.

Given a derivation $G \Rightarrow^* H$ and a node v in G , v is either removed or has a unique descendant in H . This is formalized by the following *track function*. For a direct derivation $G \Rightarrow H$, $track_{G \Rightarrow H}: V_G \rightarrow V_H$ is the partial function defined by

$$track_{G \Rightarrow H}(v) = \begin{cases} c_V(v) & \text{if } v \in D, \\ \text{undefined} & \text{otherwise} \end{cases}$$

where c is the morphism $D \rightarrow H$ in the double pushout in Figure 3.3. The track function is extended on derivations as follows: if $G \Rightarrow^* H$ by an isomorphism $i: G \rightarrow H$, then $track_{G \Rightarrow^* H} = i_V$; if $G \Rightarrow^* H$ by a sequence $G_0 \Rightarrow \dots \Rightarrow G_n$ of direct derivations, then

$$track_{G \Rightarrow^* H} = track_{G_{n-1} \Rightarrow G_n} \circ \dots \circ track_{G_0 \Rightarrow G_1}.$$

The subscript of a track function may be omitted if no ambiguity is possible.

The following theorem is essential for the proof of some confluence results in Chapter 7 and Section 8.2. It was announced in [Ros75a] and proved for the graph case by Ehrig, Kreowski, and Rosen [EK76, ER76].

Theorem 3.8 *Let $H_1 \leftarrow_{p_1, g_1} G \Rightarrow_{p_2, g_2} H_2$ be direct derivations through rules $p_i = (L_i \leftarrow K_i \rightarrow R_i)$, for $i = 1, 2$. If $g_1 L_1 \cap g_2 L_2 = g_1 K_1 \cap g_2 K_2$, then there is a hypergraph M such that $H_1 \Rightarrow_{p_2} M \leftarrow_{p_1} H_2$ and $track_{G \Rightarrow H_1 \Rightarrow M} = track_{G \Rightarrow H_2 \Rightarrow M}$.*

Proof With the following three remarks, the proof in [EK76] holds without change because it relies on abstract pushout properties.

1. Let $G \Rightarrow_{p_i, g_i} H_i$ have the following diagram, for $i = 1, 2$:

$$\begin{array}{ccccc} L_i & \longleftarrow & K_i & \longrightarrow & R_i \\ g_i \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D_i & \longrightarrow & H_i \end{array}$$

The assumption $g_1 L_1 \cap g_2 L_2 = g_1 K_1 \cap g_2 K_2$ implies $g_1 L_1 \subseteq D_2$ and $g_2 L_2 \subseteq D_1$. Hence g_1 and g_2 have restrictions $g'_1: L_1 \rightarrow D_2$ and $g'_2: L_2 \rightarrow D_1$, that is,

$$L_1 \xrightarrow{g_1} G = L_1 \xrightarrow{g'_1} D_2 \rightarrow G \text{ and } L_2 \xrightarrow{g_2} G = L_2 \xrightarrow{g'_2} D_1 \rightarrow G$$

(where juxtaposition of arrows denotes composition). Therefore the two direct derivations are *parallel independent* in the sense of [EK76].

2. The proof refers to a generalization of a “division lemma” from a previous paper of Ehrig and Kreowski. Instead, the “triple-pushout-lemma” in [EK79] (Corollary 2.3) can be used, the proof of which holds also for the hypergraph case.

3. The additional property “ $track_{G \Rightarrow H_1 \Rightarrow M} = track_{G \Rightarrow H_2 \Rightarrow M}$ ” is not stated in [EK79] but can be extracted from the proof. \square

Chapter 4

Representing Terms by Jungles

Representing functional expressions such that common subexpressions can be shared requires graph-like structures. Usually *directed acyclic graphs (dags)* are chosen for this purpose: nodes represent function symbols and edges point to arguments. Labelling nodes with function symbols, however, is problematic when graphs shall be rewritten according to the Berlin approach. Consider a function definition of the form $f(\dots) = g(\dots)$. A corresponding graph rule ($L \leftarrow K \rightarrow R$) had to transform a node v labelled with f into a node labelled with g . Because $K \rightarrow L$ and $K \rightarrow R$ are label-preserving, v cannot just be relabelled. But removing v does neither work, since then the rule were not applicable whenever v is the target of any edges. Ehrig and Rosen [ER76, ER80a] and later Padawitz [Pad82] overcame this problem by extending the Berlin approach such that graph morphisms need not be label-preserving. The drawbacks of this solution are that the theory is burdened by additional technicalities and that results of the standard approach have to be re-proved.

To avoid these drawbacks, an alternative representation of functional expressions was proposed in [Plu86]¹. The *jungles* introduced there contain function symbols as edge labels while argument and result sorts of function symbols become node labels. This representation was further improved in [HKP88, HP88a] by using hypergraphs instead of graphs. Then, after finding the left-hand pattern $f(\dots)$, the definition $f(\dots) = g(\dots)$ is executed by removing a hyperedge labelled with f and putting in a jungle representing $g(\dots)$. The details of jungle evaluation are elaborated in Chapter 5.

4.1 Jungles and Jungle Morphisms

The jungle definition given below is essentially equivalent to the one used in [HKP88, HP88a] and later jungle papers. There are two slight modifications.

¹A similar representation was also used by Hoffmann [Hof83].

Firstly, the labelling condition for jungle edges is omitted. Instead, it is generally assumed that signatures come as many-sorted declarations of function symbols and variables. Secondly, variables are represented like constants instead of as (names of) nodes. This makes it unnecessary to require that derivations preserve variable nodes.

General Assumption 4.1 From now on, for every signature Σ and each f in Σ_E , $Type_\Sigma(f)$ is a singleton $\{\langle s, w \rangle\}$ with $|s| = 1$. Moreover, it is assumed that Σ_E includes a distinguished subset Σ_X such that for each x in Σ_X , $Type_\Sigma(x)$ has the form $\{\langle s, \lambda \rangle\}$. Also, for each s in Σ_V , the set $\{x \in \Sigma_X \mid Type_\Sigma(x) = \{\langle s, \lambda \rangle\}\}$ is assumed to be nonempty. In the following Σ denotes such a signature.

The elements of Σ_V , $\Sigma_E - \Sigma_X$, and Σ_X are called *sorts*, *function symbols*, and *variables*, respectively. For a function symbol f with $Type_\Sigma(f) = \{\langle s, w \rangle\}$, the type may be referred to by writing $f:w \rightarrow s$. The string w contains the *argument sorts* of f while s is the *result sort*. If $w = \lambda$, then f is a *constant* and the type may be indicated by $f:s$. This notation is also used for variables.

Definition 4.2 (jungle) A hypergraph G over Σ is a *jungle* if

- (1) $outdegree_G(v) \leq 1$ for each $v \in V_G$, and
- (2) G is acyclic.

Nodes with outdegree 0 are called *open nodes*. Jungles without open nodes are said to be *closed*.

Example 4.3 Suppose that Σ contains sorts “nat” and “queue”, function symbols

```

0: nat
succ: nat → nat
nil: queue
add: nat queue → queue
rem: queue → queue
get: queue → nat

```

and a variable $x:queue$. Then the hypergraph in Figure 4.1 is a (closed) jungle. In pictures of jungles, the unnumbered arrows leaving a hyperedge are implicitly ordered from left to right.

The set of all jungles over Σ can be characterized by a hypergraph grammar that generates every jungle from the empty jungle (which has empty node and edge sets). See [HKP91], where from this result a structural induction principle and an efficient syntax analysis for jungles are derived. These topics are not treated in this thesis.

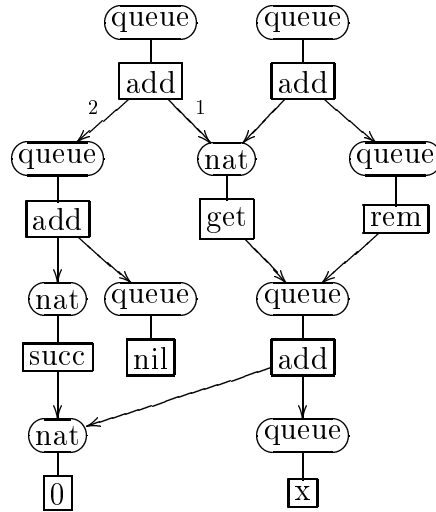


Figure 4.1: A jungle

Since jungles are acyclic, there are two induction principles for showing that all nodes of a jungle G have a property P : Using *top-down induction* one shows (1) P holds for each node with indegree 0, and (2) for each edge e , if P holds for $s_G(e)$, then P holds for each $v \in t_G(e)$. Alternatively, *bottom-up induction* proceeds by showing (1) P holds for each node with outdegree 0, and (2) for each edge e , if P holds for each $v \in t_G(e)$, then P holds for $s_G(e)$. Both techniques are frequently used in the sequel. They can be easily verified by induction on the number of nodes in a jungle.

The two conditions in Definition 4.2 ensure that each node in a closed jungle represents a unique functional expression. Let us first recall the definition of many-sorted terms. Consider a sort s in Σ_V . A *term* of sort s is a string $t \in \Sigma_E^*$ such that either $t \in \Sigma_E$ with $t: \lambda \rightarrow s$, or t has the form $f t_1 \dots t_n$ with $f: s_1 \dots s_n \rightarrow s$ and t_i is a term of sort s_i for $i = 1, \dots, n$. $T_s(\Sigma)$ denotes the set of all terms of sort s and $T(\Sigma) = \bigcup_{s \in \Sigma_V} T_s(\Sigma)$ is the set of all terms over Σ . For $t, u \in T(\Sigma)$, u is a *subterm* of t if $u = t$ or $t = f t_1 \dots t_n$ and u is a subterm of some t_i , $1 \leq i \leq n$. Note that the decomposition $f t_1 \dots t_n$ of t is unique (see [Coh81] for a proof).

Now, given a closed jungle and some node v in it, a term can be assigned to v in a straightforward way: starting with the edge outgoing from v one descends along the hyperedges and collects their labels, analogous to a preorder tree traversal. This procedure terminates because jungles are acyclic, and yields a unique term since all nodes have outdegree one.

Definition 4.4 (term representation) Let G be a closed jungle. Then the function $term_G: V_G \rightarrow T(\Sigma)$ is defined by

$$term_G(v) = m_G(e) term_G^*(t_G(e)),$$

where e is the unique edge with $s_G(e) = v$. The set $\{term_G(v) \mid v \in V_G\}$ of all terms represented by G is denoted by $TERM_G$.

The well-definedness of $term_G$ can be easily shown by bottom-up induction. It should be noticed that each node v is labelled with the sort of $term_G(v)$.

As an example, the left one of the two topmost nodes in Figure 4.1 represents the term $\text{add}(\text{get}(\text{add}(0, x)), \text{add}(\text{succ}(0), \text{nil}))$. To enhance readability, here and in later examples terms are written with parentheses and commas.

The following two lemmas about the representation of terms are frequently used in the sequel. Both have simple proofs by bottom-up induction.

Lemma 4.5 *Let G be a closed jungle, v be a node in G , and t be a term. Then t is a subterm of $term_G(v)$ if and only if there is a node v' such that $term_G(v') = t$ and $v \geq_G v'$.*

Hypergraph morphisms between jungles are henceforth called *jungle morphisms*. The next lemma shows that jungle morphisms are compatible with the representation of terms.

Lemma 4.6 *For every jungle morphism $f: G \rightarrow H$ between closed jungles,*

$$term_H \circ f_V = term_G.$$

While equations and term rewrite rules have “formal parameters” in form of variables, open nodes serve this purpose in hypergraph rules with jungles. In translating rewrite rules into jungle evaluation rules, variables have to be converted into open nodes. This is simply achieved by cutting of Σ_X -labelled edges in closed jungles.

Definition 4.7 For a closed jungle G , \underline{G} is the subjungle² of G with node set V_G and edge set $\{e \in E_G \mid m_G(e) \notin \Sigma_X\}$.

When term rewrite rules are applied to terms, variables are instantiated by means of substitutions. The corresponding concept for jungle evaluation is the instantiation of open nodes by hypergraph morphisms. It turns out that jungle morphisms induce substitutions, provided that open nodes correspond uniquely to variables. This can be enforced by using a special kind of jungles.

²Subhypergraphs of jungles (which are again jungles) are called *subjungles*.

Definition 4.8 (variable-collapsed jungle) A closed jungle G is *variable-collapsed* if for all $v, v' \in V_G$, $\text{term}_G(v) = \text{term}_G(v') \in \Sigma_X$ implies $v = v'$.

Recall that a *substitution* $\sigma: \Sigma_X \rightarrow T(\Sigma)$ is a mapping that assigns to each variable a term of the same sort. The extension of σ on terms is also denoted by σ ; it is the identity on constants and maps composite terms $f t_1 \dots t_n$ to $f \sigma(t_1) \dots \sigma(t_n)$.

Definition 4.9 (induced substitution) Let G be a variable-collapsed jungle, H be a closed jungle, and $f: \underline{G} \rightarrow H$ be a jungle morphism. Then the *induced substitution* $\sigma_f: \Sigma_X \rightarrow T(\Sigma)$ is defined by

$$\sigma_f(x) = \begin{cases} \text{term}_H(f_V(s_G(e))) & \text{if } x = m_G(e) \text{ for some } e \in E_G, \\ x & \text{otherwise.} \end{cases}$$

Note that σ_f is well-defined because f_V is sort preserving.

Lemma 4.10 *With f and σ_f as in Definition 4.9, the following diagram is commutative:*

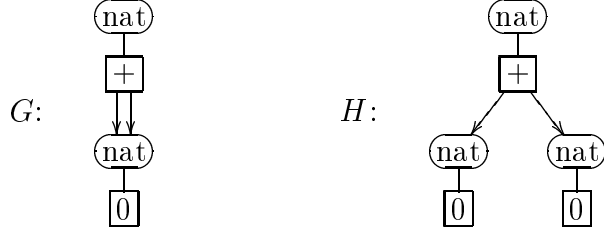
$$\begin{array}{ccc} V_G & \xrightarrow{f_V} & V_H \\ \text{term}_G \downarrow & & \downarrow \text{term}_H \\ T(\Sigma) & \xrightarrow{\sigma_f} & T(\Sigma) \end{array}$$

Proof By bottom-up induction in G . Let $e \in E_G$ such that $\sigma_f(\text{term}_G(v)) = \text{term}_H(f_V(v))$ for each $v \in t_G(e)$. If $m_G(e) \in \Sigma_X$, then $t_G(e) = \lambda$ and hence $\sigma_f(\text{term}_G(s_G(e))) = \sigma_f(m_G(e)) = \text{term}_H(f_V(s_G(e)))$. If $m_G(e) \notin \Sigma_X$, then

$$\begin{aligned} \sigma_f(\text{term}_G(s_G(e))) &= \sigma_f(m_G(e) \text{term}_G^*(t_G(e))) \\ &= m_G(e) \sigma_f^*(\text{term}_G^*(t_G(e))) \\ &= m_G(e) \text{term}_H^*(f_V^*(t_G(e))) && \text{ind. hypothesis} \\ &= m_H(f_E(e)) \text{term}_H^*(t_H(f_E(e))) \\ &= \text{term}_H(s_H(f_E(e))) \\ &= \text{term}_H(f_V(s_G(e))). \end{aligned}$$

□

Lemma 4.6 shows that jungle morphisms between closed jungles preserve represented terms. So the existence of such a morphism $f: G \rightarrow H$ implies $\text{TERM}_G \subseteq \text{TERM}_H$. To see that the converse needs not hold, consider a function symbol $+: \text{nat nat} \rightarrow \text{nat}$ and the following jungles:



There is no jungle morphism $G \rightarrow H$ although $TERM_G = TERM_H$. The reason is that H is “less collapsed” than G . This suggests to consider jungles with a maximum degree of sharing, that is, jungles in which each two different nodes represent different terms.

Definition 4.11 (fully collapsed jungle) A closed jungle G is *fully collapsed* if $term_G$ is injective.

In the next section it is shown that every closed jungle can be transformed into a fully collapsed jungle representing the same terms.

Lemma 4.12 (existence of jungle morphisms) *Let G and H be closed jungles such that $TERM_G \subseteq TERM_H$. If H is fully collapsed, then there is a unique jungle morphism $f: G \rightarrow H$.*

Proof By assumption, for each $v \in V_G$ there is $v' \in V_H$ such that $term_H(v') = term_G(v)$. The node v' is uniquely determined because H is fully collapsed. That is, there is a unique function $f_V: V_G \rightarrow V_H$ such that

$$term_H \circ f_V = term_G. \quad (4.1)$$

Then $l_H \circ f_V = l_G$ follows from the fact that each node is labelled with the (unique) sort of its associated term.

Now consider some $e \in E_G$. By (4.1), $term_H(f_V(s_G(e))) = term_G(s_G(e)) = m_G(e) term_G^*(t_G(e))$ and hence there is a unique $e' \in E_H$ with $s_H(e') = f_V(s_G(e))$ and $m_H(e') = m_G(e)$. In other words, there is a unique function $f_E: E_G \rightarrow E_H$ such that

$$s_H \circ f_E = f_V^* \circ s_G \quad (4.2)$$

and $m_H \circ f_E = m_G$. To complete the proof that $\langle f_V, f_E \rangle$ is a jungle morphism, $t_H \circ f_E = f_V^* \circ t_G$ must be shown. This equality follows from the computation

$$\begin{aligned} m_H(f_E(e)) term_H^*(t_H(f_E(e))) &= term_H(s_H(f_E(e))) \\ &= term_H(f_V^*(s_G(e))) & (4.2) \\ &= term_G(s_G(e)) & (4.1) \\ &= m_G(e) term_G^*(t_G(e)) \\ &= m_G(e) term_H^*(f_V^*(t_G(e))) & (4.1) \end{aligned}$$

by injectivity of $term_H$. So $f = \langle f_V, f_E \rangle$ is a jungle morphism. f_V is uniquely determined since (4.1) is a necessary condition by Lemma 4.6. Then the morphism property (4.2) leaves only one choice for f_E . \square

As a consequence of Lemma 4.12, fully collapsed jungles are determined uniquely up to isomorphism by the terms they represent.

Theorem 4.13 (uniqueness of fully collapsed jungles) *Two fully collapsed jungles G and H are isomorphic if and only if $TERM_G = TERM_H$.*

Proof Isomorphic closed jungles clearly represent the same sets of terms. Conversely, assume that $TERM_G = TERM_H$. Then, by Lemma 4.12, there are jungle morphisms $f:G \rightarrow H$ and $g:H \rightarrow G$ such that $term_H \circ f_V = term_G$ and $term_G \circ g_V = term_H$. Hence $term_G \circ g_V \circ f_V = term_G$, which by injectivity of $term_G$ implies $g_V \circ f_V = id_{V_G}$ (where id_{V_G} is the identity on V_G). Analogously $f_V \circ g_V = id_{V_H}$ is obtained, so f_V and g_V are bijections. Then f_E and g_E are also injective. Surjectivity holds because f_V and g_V are surjective and preserve represented terms. Thus f and g are isomorphisms. \square

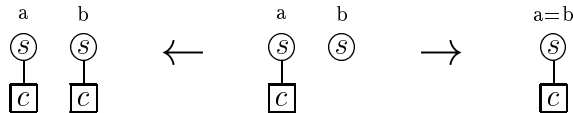
4.2 Folding

For every closed jungle G a fully collapsed jungle \tilde{G} can be constructed such that $TERM_{\tilde{G}} = TERM_G$ (where $TERM_G$ can be used as node and edge set for \tilde{G} , see [HP88b]). Hence, by Theorem 4.13, the set of all jungles that represent the same terms as G contains \tilde{G} as a (up to isomorphism) unique representative. Moreover, it is clear that \tilde{G} has the least number of nodes and edges among all these jungles. Because of these properties, and not to forget Lemma 4.12, fully collapsed jungles deserve special attention.

This section is concerned with hypergraph rules that allow to transform jungles into their fully collapsed counterparts. In applying such a rule, two edges that have the same labels and target nodes are “folded”, that is, identified. Repeated rule applications stepwise increase the degree of sharing and eventually yield a fully collapsed jungle.

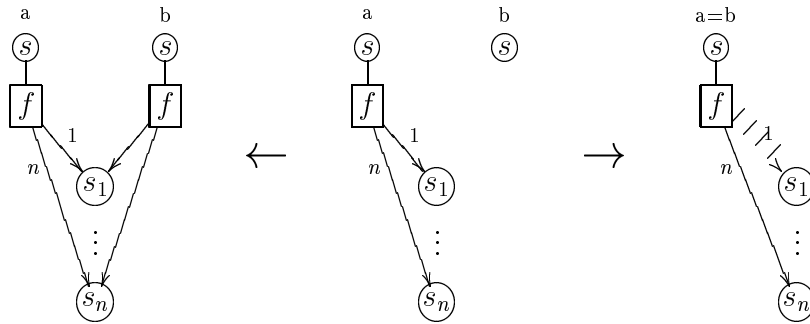
Definition 4.14 (folding rules) The following rules constitute the set \mathcal{F} of *folding rules*³.

- For each constant and each variable $c: s$ in Σ :



³In pictures of hypergraph rules, letters a,b,c,... are used as node names to represent morphisms uniquely. Here “a=b” means that the nodes a and b are identified.

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



For every closed jungle G , a direct derivation $G \Rightarrow_{\mathcal{F}} H$ is a *folding step* and a derivation $G \Rightarrow_{\mathcal{F}}^* H$ is a *folding*.

It is important to notice that for every folding step $G \Rightarrow_{p,g} H$, the morphism g does not identify the two edges in the left-hand side of p . This is prevented by the identification condition for direct derivations, since only one of the edges appears in the interface of p . As a consequence, $G \Rightarrow_{p,g} H$ decreases the number of edges in G by one. So $\Rightarrow_{\mathcal{F}}$ is a terminating relation.

Example 4.15 Figure 4.2 shows a folding step with jungles over the signature of Example 4.3.

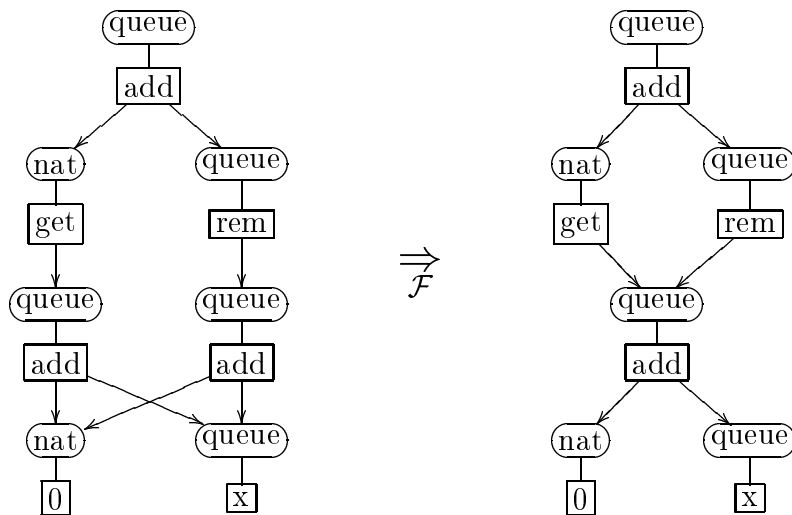


Figure 4.2: A folding step

Lemma 4.16 *Let G be a closed jungle and $G \Rightarrow_{\mathcal{F}} H$ be a folding step. Then*

- (1) H is a closed jungle,
- (2) there is a surjective jungle morphism $f: G \rightarrow H$ with $f_V = \text{track}_{G \Rightarrow H}$,
- (3) $\text{term}_H \circ \text{track}_{G \Rightarrow H} = \text{term}_G$ and $\text{TERM}_H = \text{TERM}_G$.

Proof (1) Let $G \Rightarrow_{\mathcal{F}} H$ be given by the following double pushout:

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ g \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \xrightarrow{c} & H \end{array}$$

Let e_1, e_2 be the two edges in gL , with $e_2 \in gL - gK$. According to the construction of direct derivations in Section 3.3, D is obtained from G by removing e_2 . So D is a jungle with a unique open node $o = s_G(e_2)$. Let $v = s_G(e_1)$. Then H is obtained from D by identifying v and o . So each node in H has a unique outgoing edge. To see that H is acyclic, suppose the contrary. This is only possible if $c_V(o) >_H c_V(o)$, since D is acyclic. Then $v >_D o$ must hold because o is an open node. Hence $v >_G o$. But the structure of L ensures that for each node x in G , $v >_G x$ implies $o >_G x$. Thus in particular $o >_G o$, contradicting the fact that G is acyclic.

(2) Define mappings $f_V: V_G \rightarrow V_H$ and $f_E: E_G \rightarrow E_H$ by $f_V = c_V$ and $f_E(e) = \text{if } e = e_2 \text{ then } c_E(e_1) \text{ else } c_E(e)$. Then $f = \langle f_V, f_E \rangle$ is a jungle morphism: e_1 and e_2 have the same label and the same target string, and their source nodes are identified by f_V . f is surjective because $K \rightarrow R$ and c are so.

(3) By Lemma 4.6, $\text{term}_H \circ f_V = \text{term}_G$ for the morphism f defined above. It follows $\text{term}_H \circ \text{track}_{G \Rightarrow H} = \text{term}_G$ since $f_V = c_V = \text{track}_{G \Rightarrow H}$. Then $\text{TERM}_H = \text{TERM}_G$ by surjectivity of f . \square

By point (1) of the above lemma, folding steps preserve closed jungles. If not stated otherwise, $\Rightarrow_{\mathcal{F}}$ is henceforth considered as a relation on (isomorphism classes of) closed jungles. The next lemma shows that, as expected, the normal forms of $\Rightarrow_{\mathcal{F}}$ are just the fully collapsed jungles.

Lemma 4.17 (characterization of fully collapsed jungles) *A closed jungle is fully collapsed if and only if no folding rule is applicable to it.*

Proof Let G be a fully collapsed jungle and $f: L \rightarrow G$ be a jungle morphism, where L is the left-hand side of some folding rule. Let v_1, v_2 be the source nodes of the two edges in L . By the structure of L , $\text{term}_G(f_V(v_1)) = \text{term}_G(f_V(v_2))$

and hence $f_V(v_1) = f_V(v_2)$. Then f identifies also the two edges in L , but this violates the identification condition. Therefore no folding rule is applicable to G .

Conversely, let G be a closed jungle that is not fully collapsed. Then there are two distinct nodes v_1, v_2 in G with $\text{term}_G(v_1) = \text{term}_G(v_2)$. In particular, the pair $\langle v_1, v_2 \rangle$ can be chosen such that it is minimal with respect to $>_G$. That is, for each pair $\langle v'_1, v'_2 \rangle$ of distinct nodes that represent the same term, $v_1 \not\prec_G v'_1$ and $v_2 \not\prec_G v'_2$. Then the edges outgoing from v_1 and v_2 have the same label and the same target string. Hence a folding rule can be applied. \square

Since $\Rightarrow_{\mathcal{F}}$ is terminating, Lemma 4.17 implies that every closed jungle can be transformed into a fully collapsed jungle by applying folding rules as long as possible. This nondeterministic procedure yields a unique jungle, as the next result shows.

Theorem 4.18 *The relation $\Rightarrow_{\mathcal{F}}$ is confluent.*

Proof Let $H_1 \Leftarrow_{\mathcal{F}}^* G \Rightarrow_{\mathcal{F}}^* H_2$ be foldings. Because $\Rightarrow_{\mathcal{F}}$ is terminating, there are foldings $H_i \Rightarrow_{\mathcal{F}}^* X_i$ ($i = 1, 2$) such that no folding rule is applicable to X_1 and X_2 . By Lemma 4.17 and Lemma 4.16(3), X_1 and X_2 are fully collapsed and represent $TERM_G$. Hence $X_1 \cong X_2$ by Theorem 4.13. \square

This theorem can be strengthened in that $\Rightarrow_{\mathcal{F}}$ is even strongly confluent, see [HP88b], but the stronger property is not needed in the following chapters.

Chapter 5

Jungle Evaluation

With the preparation of the preceding chapter, hypergraph rewriting can now be applied to the evaluation of functional expressions. In this chapter function definitions in form of term rewriting systems are translated into systems of jungle rewrite rules. The idea is to apply these rules together with the folding rules of Section 4.2 in order to evaluate the terms represented by jungles. The resulting computational model is called *jungle evaluation*. It is shown that jungle evaluation is sound for function evaluation in the same sense as term rewriting is. But the sharing of common subterms makes the behaviour of jungle evaluation different from that of term rewriting. In particular, the two models differ with respect to termination and confluence.

5.1 Equations and Term Rewrite Rules

This section reviews basic notions and a principal result concerning equations and term rewrite rules. Comprehensive surveys on term rewriting systems are given by Huet and Oppen [HO80], Dershowitz and Jouannaud [DJ90], and Klop [Klo92].

A Σ -algebra A consists of non-empty sets¹ A_s for all sorts s in Σ , distinguished elements $c_A \in A_s$ for all constants $c: s$ in Σ , and functions $f_A: A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$ for all function symbols $f: s_1 \dots s_n \rightarrow s$ ($n \geq 1$) in Σ .

Given a Σ -algebra A , a *valuation* $v: \Sigma_X \rightarrow A$ assigns to each variable $x: s$ an element in A_s (so v is a “sort indexed” family of mappings). The extension of v on terms is also denoted by v and maps constants c to c_A and composite terms $f t_1 \dots t_n$ to $f_A(v(t_1), \dots, v(t_n))$.

An *equation* over Σ is a pair of terms l, r of the same sort, written $l \doteq r$. $l \doteq r$ is *valid* in a Σ -algebra A , denoted by $A \models l \doteq r$, if $v(l) = v(r)$ for all valuations v .

¹Restricting to nonempty sets avoids problems with the soundness and completeness of equational deduction; see [HO80] for the definition of “sensible” signatures which exclude the critical cases of empty domains.

That is, the variables in l and r are considered as universally quantified. Validity is extended to classes of algebras in the obvious way: $C \models l \doteq r$ holds for a class of algebras C if $l \doteq r$ is valid in each member of C . Given a set E of equations, an algebra A is a *model* of E , denoted by $A \models E$, if all equations in E are valid in A . $Mod(E)$ denotes the class of all models of E .

Example 5.1 With Σ as in Example 4.3 and variables $m, n: \text{nat}, q: \text{queue}$ one may have the following equations:

$$\begin{aligned} \text{rem}(\text{nil}) &\doteq \text{nil} \\ \text{rem}(\text{add}(n, \text{nil})) &\doteq \text{nil} \\ \text{rem}(\text{add}(m, \text{add}(n, q))) &\doteq \text{add}(m, \text{rem}(\text{add}(n, q))) \\ \text{get}(\text{add}(n, \text{nil})) &\doteq n \\ \text{get}(\text{add}(m, \text{add}(n, q))) &\doteq \text{get}(\text{add}(n, q)) \end{aligned}$$

Then $\text{get}(\text{add}(m, \text{add}(n, q))) \doteq \text{get}(\text{add}(\text{get}(\text{add}(n, q)), \text{nil}))$ is valid in all models of these equations.

An equation $l \doteq r$ can also be used as a left-to-right *rewrite rule*; in this case the notation $l \rightarrow r$ is used and two restrictions are imposed²:

- Each variable that occurs in r occurs also in l .
- l is not a variable.

Given a set P of rewrite rules, $\mathcal{R} = \langle \Sigma, P \rangle$ is a *term rewriting system*. From now on \mathcal{R} denotes an arbitrary term rewriting system, if not stated otherwise.

Let Σ be extended by constants \square_s , for all sorts s . Then a *context* is a term *con* with a unique occurrence of one of these constants. For a Σ -term t of sort s , $\text{con}[t]$ is the term that is obtained by replacing \square_s in *con* by t .

Now the *rewrite relation* $\rightarrow_{\mathcal{R}}$ on $T(\Sigma)$ is defined by: $t \rightarrow_{\mathcal{R}} u$ if there is a rule $l \rightarrow r$ in \mathcal{R} , a substitution σ , and a context *con* such that $t = \text{con}[\sigma(l)]$ and $u = \text{con}[\sigma(r)]$. \mathcal{R} is said to be terminating, confluent, etc. if $\rightarrow_{\mathcal{R}}$ has the respective property (see Chapter 2).

By considering \mathcal{R} as a set of equations, $Mod(\mathcal{R})$ provides \mathcal{R} with a semantics in the sense of logic. A fundamental property of term rewriting is that the equivalence generated by $\rightarrow_{\mathcal{R}}$ is sound and complete with respect to validity in $Mod(\mathcal{R})$:

Theorem 5.2 For arbitrary terms t and u ,

$$Mod(\mathcal{R}) \models t \doteq u \text{ if and only if } t \xrightarrow[\mathcal{R}]^* u.$$

²The first condition ensures that rewrite rules can be applied with substitutions that are found by pattern matching, and violation of the second condition causes non-termination. Moreover, the translation of rewrite rules into jungle rules considered in the next section had to be more complicated without these conditions.

This result follows from the well-known completeness theorem of Birkhoff [Bir35] for equational deduction (see for example [SA91]).

Assume, for instance, that \mathcal{R} contains the five equations of Example 5.1, considered as left-to-right rewrite rules. Then

$$\begin{array}{ccc}
 \text{get}(\text{add}(\text{m}, \text{add}(\text{n}, \text{q}))) & & \\
 & \searrow_{\mathcal{R}} & \\
 & & \text{get}(\text{add}(\text{n}, \text{q})) \\
 & \nearrow_{\mathcal{R}} & \\
 \text{get}(\text{add}(\text{get}(\text{add}(\text{n}, \text{q})), \text{nil})) & &
 \end{array}$$

proves the validity of $\text{get}(\text{add}(\text{m}, \text{add}(\text{n}, \text{q}))) \doteq \text{get}(\text{add}(\text{get}(\text{add}(\text{n}, \text{q})), \text{nil}))$ in $\text{Mod}(\mathcal{R})$.

5.2 Evaluation Rules

The goal of this section is to construct hypergraph rules such that jungle rewriting with these rules performs function evaluation in a similar way as term rewriting does. This is achieved by translating term rewrite rules into jungle rewrite rules which perform term rewrite steps on the terms represented by jungles. Then the soundness of the translated rules for function evaluation is a consequence of the soundness of term rewriting (in the sense of Theorem 5.2).

In designing a jungle rule ($L \leftarrow K \rightarrow R$) for a term rewrite rule $l \rightarrow r$ one encounters the problem that an instance $\sigma(l)$ of the left-hand side in a jungle may contain shared subterms; that is, nodes representing subterms of $\sigma(l)$ may be target nodes of edges “outside” of $\sigma(l)$. As a consequence, ($L \leftarrow K \rightarrow R$) has to preserve all proper subterms of $\sigma(l)$. This is realized by letting L represent l and by choosing K as the subjungle of L that contains all nodes and edges except the edge e labelled with the leftmost function symbol in l . Then R is obtained by gluing K with a jungle R' representing r such that the “root” of R' is fused with the source node of e , and edges labelled with variables are fused with the corresponding edges in K . Finally, all variable edges in L , K , and R are cut off. The resulting rule ($\underline{L} \leftarrow \underline{K} \rightarrow \underline{R}$) is schematically depicted in Figure 5.1 (where $l = f t_1 \dots t_n$ for a function symbol f and terms t_1, \dots, t_n).

The above considerations aim at making ($\underline{L} \leftarrow \underline{K} \rightarrow \underline{R}$) applicable to a jungle G whenever the underlying rewrite rule $l \rightarrow r$ is applicable to a term in TERM_G . This requires also to choose L as little collapsed as possible, in order to guarantee the existence of a jungle morphism $\underline{L} \rightarrow G$ independently of the degree of collapsing in G . (How far this can be realized is considered in the next section.) The special kind of jungles needed for the representation of l is introduced in the following definition.

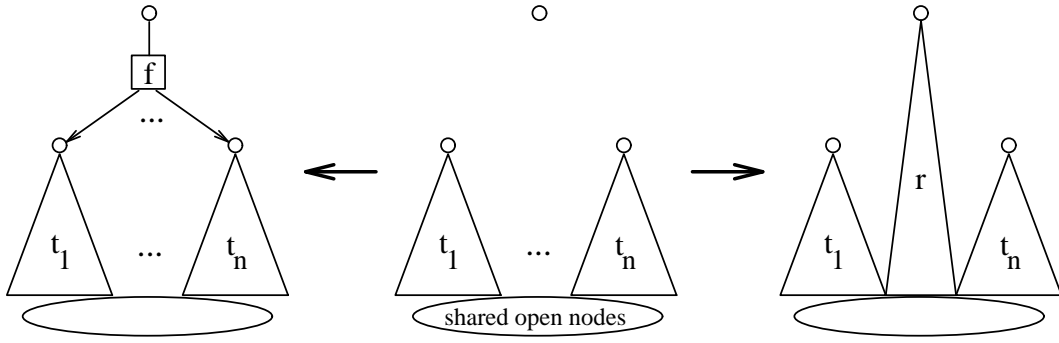


Figure 5.1: Evaluation rule (schematic)

Definition 5.3 (tree with shared variables) A variable-collapsed jungle T is a tree with shared variables if there is a unique node $root_T$ with indegree 0, and if each node with an indegree greater than 1 represents a variable.

Given a term t , $\diamond t$ is a tree with shared variables such that $term_{\diamond t}(root_{\diamond t}) = t$. $\diamond t$ can always be constructed and is unique up to isomorphism, see [HKP91]³.

Definition 5.4 (evaluation rules) Given a term rewrite rule $l \rightarrow r$, construct a hypergraph rule $(L \leftarrow K \rightarrow R)$ as follows:

- (1) $L = \diamond l$.
- (2) K is obtained from L by removing the unique edge outgoing from $root_{\diamond l}$, and $K \rightarrow L$ is the inclusion of K in L .
- (3) R is obtained by gluing K and $\diamond r$ such that (i) $root_{\diamond l}$ is identified with $root_{\diamond r}$, and (ii) for each pair $\langle e, e' \rangle \in E_{\diamond l} \times E_{\diamond r}$ with $m_{\diamond l}(e) = m_{\diamond r}(e') \in \Sigma_X$, e is identified with e' (meaning that their source nodes are also identified). To be more precise, let I be the subjungle of K consisting of $root_{\diamond l}$ and all edges labelled with variables occurring in r (together with their source nodes). Let $I \rightarrow K$ be the associated inclusion, and let $I \rightarrow \diamond r$ map $root_{\diamond l}$ to $root_{\diamond r}$ and each edge to its unique counterpart. Then R and the morphism $K \rightarrow R$ are obtained by the following hypergraph pushout:

$$\begin{array}{ccc}
 I & \longrightarrow & \diamond r \\
 \downarrow & & \downarrow \\
 K & \longrightarrow & R
 \end{array}$$

³Trees with shared variables are called “variable-collapsed trees” in [HKP91]. Here a different notion is used to avoid confusion with the “collapsed trees” introduced in Chapter 6.

Now $(\underline{L} \leftarrow \underline{K} \rightarrow \underline{R})$ is the *evaluation rule* for $l \rightarrow r$, where the morphisms are the restrictions of $K \rightarrow L$ and $K \rightarrow R$.

\mathcal{E} is the set of all evaluation rules for rewrite rules in \mathcal{R} . For every closed jungle G , a direct derivation $G \Rightarrow_{\mathcal{E}} H$ is an *evaluation step*. If $G \Rightarrow_{p,g} H$ such that p is the evaluation rule for a rewrite rule $l \rightarrow r$, then the evaluation step is said to be *based on* $l \rightarrow r$.

The above choice for the structure of the right-hand side R is not the only possible; for example, R is fully collapsed in [HP88a]. The view taken here (and in [HP91]) is that evaluation steps should introduce only as much sharing as necessary. Then different strategies for applying folding rules between evaluation steps can be chosen to control the degree of sharing.

With a similar intention, Farmer and Watro [FW90, FW91] show the soundness of arbitrary “structure sharing schemes” for function evaluation. They allow evaluation steps that add arbitrary structure to a graph, if only the represented terms behave properly with respect to term rewriting. Such a general model of computation, however, lacks the essential feature of (hyper-)graph rewriting that a single rewrite step requires only constant space.

Example 5.5 Figure 5.2 shows the evaluation rules for the rewrite rules

$$\begin{aligned} x \times (y + z) &\rightarrow (x \times y) + (x \times z), \\ x + 0 &\rightarrow x, \end{aligned}$$

where \times and $+$ are binary function symbols and 0 is a constant⁴. Note that in the second evaluation rule, the root of the left-hand side is identified with the open node that corresponds to the variable x . In Figure 5.3 an evaluation step through the first rule is shown (with an occurrence of the left-hand side in which the nodes b and c are identified).

The following lemma implies that an evaluation step identifies nodes in a jungle only if the right-hand side r of the underlying rewrite rule $l \rightarrow r$ is a variable.

Lemma 5.6 *Given an evaluation rule $(\underline{L} \leftarrow \underline{K} \xrightarrow{b} \underline{R})$ for a rewrite rule $l \rightarrow r$, the following are equivalent:*

- (1) r is not a variable.
- (2) b is injective.
- (3) b is not surjective.

⁴Here and in all later examples, the symbols x, y, z stand for variables, all terms have the same sort, and infix notation is used when convenient. Moreover, jungle nodes are depicted as unlabelled circles.

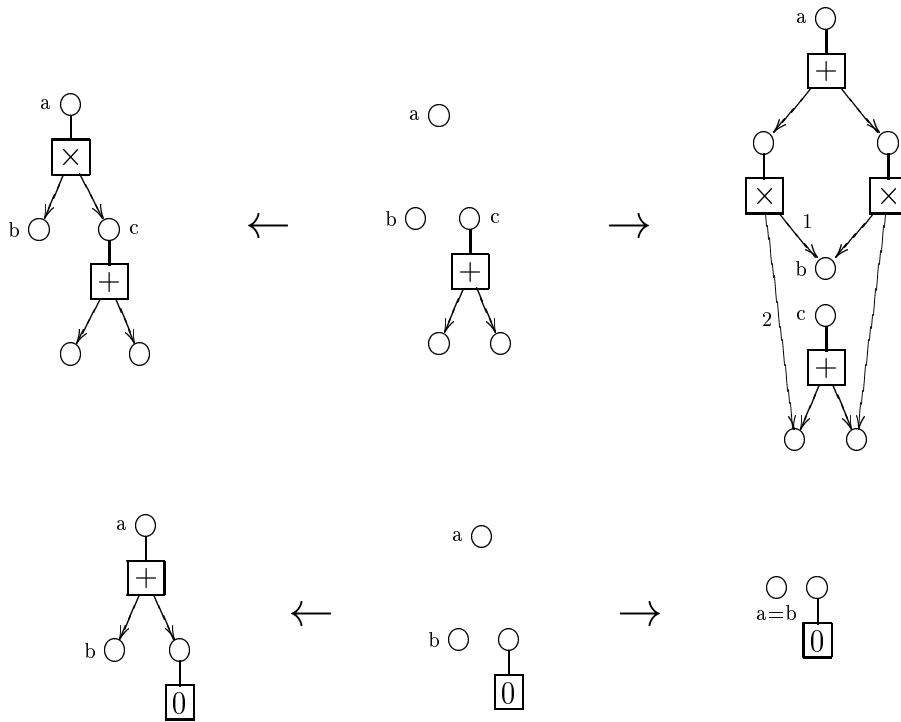


Figure 5.2: Two evaluation rules

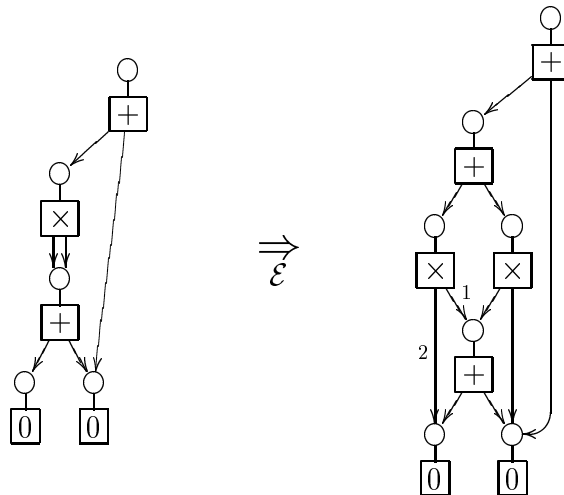


Figure 5.3: An evaluation step

Proof (1) \Rightarrow (2): If r is not a variable, then the edge in $\diamond r$ outgoing from $root_{\diamond r}$ is not labelled with a variable. So the morphism $I \rightarrow \diamond r$ in Definition 5.4 is injective, implying that $K \rightarrow R$ and b are also injective.

(2) \Rightarrow (3): If b is injective, then $b_V(root_{\diamond l})$ is the source node of an edge that has no preimage in \underline{K} (since $root_{\diamond l}$ is an open node in \underline{K}).

(3) \Rightarrow (1): If b is not surjective, then neither are $K \rightarrow R$ and $I \rightarrow \diamond r$. So $\diamond r$ is not just a single edge with its source node, which would be the case if r were a variable. \square

Lemma 5.7 *For every evaluation step $G \Rightarrow_{\varepsilon} H$, H is a closed jungle.*

Proof Let the evaluation step be given by the following double pushout:

$$\begin{array}{ccccc} \underline{L} & \longleftarrow & \underline{K} & \xrightarrow{b} & \underline{R} \\ g \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \xrightarrow{c} & H \end{array}$$

D is a subjungle of G and has $g_V(root_{\diamond l})$ as a unique open node (let $l \rightarrow r$ be the rewrite rule underlying $(\underline{L} \leftarrow \underline{K} \rightarrow \underline{R})$). If $b_V(root_{\diamond l})$ is an open node, then there is $x \in V_{\underline{K}}$ with $b_V(x) = b_V(root_{\diamond l})$ and $x \neq root_{\diamond l}$ (since l is not a variable); then $g_V(x)$ has an outgoing edge and so has $c_V(g_V(x)) = c_V(g_V(root_{\diamond l}))$. Also, there is no open node in $\underline{R} - b(\underline{K})$. It follows that each node in H has an outgoing edge; there is exactly one such edge as $b_V(root_{\diamond l})$ is the only node in $b(\underline{K})$ that may have an outgoing edge in $\underline{R} - b(\underline{K})$ (and in this case b is injective).

It remains to show that H is acyclic. To this end, suppose the contrary. Then, since D and \underline{R} are acyclic, $c_V(g_V(root_{\diamond l})) >_H c_V(g_V(root_{\diamond l}))$ must hold. This is only possible if there is an open node x in \underline{L} such that $g_V(x) >_D g_V(root_{\diamond l})$. But then $g_V(x) >_G g_V(root_{\diamond l})$, which implies $g_V(x) >_G g_V(x)$ by the fact that $root_{\diamond l} >_L x$. Hence there is a contradiction to the assumption that G is a jungle. \square

As in the case of folding steps, $\Rightarrow_{\varepsilon}$ is from now on considered as a relation on (isomorphism classes of) closed jungles, if not stated otherwise.

5.3 Soundness

The main result of this section is the *Soundness Theorem*. It states precisely the effect of an evaluation step on the terms represented by a jungle, which amounts to a parallel application of a rewrite rule to certain equal subterms. So jungle evaluation is sound with respect to term rewriting and, as a consequence, with respect to equational validity in the sense of Theorem 5.2. Moreover, this result

explains why jungle evaluation can be more efficient in time than (sequential) term rewriting.

First of all, some notions are needed for relating positions in terms with nodes in jungles. Let t be a term. The set $Pos(t)$ of *positions* in t is the subset of \mathbb{N}^* defined by

- (1) $Pos(t) = \{\lambda\}$ if t is a variable or a constant,
- (2) $Pos(t) = \{\lambda\} \cup \{i.\pi \mid 1 \leq i \leq n \text{ and } \pi \in Pos(t_i)\}$ if t is a composite term $f t_1 \dots t_n$.

Here $i.\pi$ denotes the left addition of i to the string π .

Given $\pi \in Pos(t)$, the subterm t/π is defined by

$$t/\pi = \begin{cases} t & \text{if } \pi = \lambda, \\ t_i/\rho & \text{if } \pi = i.\rho \text{ and } t = f t_1 \dots t_n. \end{cases}$$

Similar to the specification of subterms by positions, jungle nodes determine subjungles “below” these nodes.

Definition 5.8 (collapsed tree at a node) Given a closed jungle G and a node v in G , the *collapsed tree at v* , denoted by G/v , is the subjungle of G induced by $\{v' \mid v \geq_G v'\}$.

Each position in a term represented by a node v corresponds to a node in the collapsed tree at v .

Definition 5.9 (from positions to nodes) Let G be a closed jungle. Then for each $v \in V_G$ the mapping $node_{G/v}: Pos(term_G(v)) \rightarrow V_{G/v}$ is defined by $node_{G/v}(\lambda) = v$ and $node_{G/v}(i.\pi) = node_{G/v_i}(\pi)$, where $v_1 \dots v_n$ is the target string of the edge with source v .

Lemma 5.10 *Let G be a closed jungle. Then for each node v and each position π in $term_G(v)$, $term_G(v)/\pi = term_G(node_{G/v}(\pi))$.*

Proof By bottom-up induction. Let $v = s_G(e)$ for some edge e such that the proposition holds for each node in $t_G(e)$. If $\pi = \lambda$, then the proposition holds obviously for v . If $\pi = i.\rho$ for some $i \geq 1$ and $\rho \in Pos(term_G(v)/i)$, then $t_G(e) = v_1 \dots v_n$ for some $n \geq i$. Hence

$$\begin{aligned} term_G(v)/i.\rho &= term_G(v_i)/\rho \\ &= term_G(node_{G/v_i}(\rho)) \quad \text{induction hypothesis} \\ &= term_G(node_{G/v}(i.\rho)). \end{aligned}$$

□

The following two properties of evaluation rules are needed to prove the Soundness Theorem.

Lemma 5.11 *Let $(\underline{L} \leftarrow \underline{K} \xrightarrow{b} \underline{R})$ be an evaluation rule for a rewrite rule $l \rightarrow r$. Then*

- (1) $term_R(b_V(\text{root}_{\diamond l})) = r$ and
- (2) for each open node o in \underline{L} , $term_R(b_V(o)) = term_L(o)$.

Proof In the construction of R in Definition 5.4, the morphisms $K \rightarrow R$ and $\diamond r \rightarrow R$ map $\text{root}_{\diamond l}$ and $\text{root}_{\diamond r}$ to the same node; hence, since $\diamond r \rightarrow R$ preserves represented terms, $\text{root}_{\diamond l}$ is mapped by $K \rightarrow R$ to a node representing r . Moreover, $K \rightarrow R$ preserves variables (as every jungle morphism). Then (1) and (2) hold because b_V is the node mapping of $K \rightarrow R$. \square

For stating the Soundness Theorem, a special kind of parallel term rewriting is introduced which replaces a set of equal subterms simultaneously.

Definition 5.12 (\Rightarrow_{Δ}) Let t be a term, $l \rightarrow r$ be a rule in \mathcal{R} , σ be a substitution, and $\Delta \subseteq \text{Pos}(t)$ such that $t/\pi = \sigma(l)$ for each $\pi \in \Delta$. Then $t \Rightarrow_{\Delta} u$ for the term u that is obtained from t by replacing $\sigma(l)$ by $\sigma(r)$ at all positions in Δ .

Note that for every step $t \Rightarrow_{\Delta} u$ and every partition $\Delta = \Gamma \cup \Lambda$ with $\Gamma \cap \Lambda = \emptyset$, there is a term t' such that $t \Rightarrow_{\Gamma} t' \Rightarrow_{\Lambda} u$.

Example 5.13 Given a rewrite rule $f(x) \rightarrow a$, then

$$g(g(f(a), f(a)), g(f(b), f(a))) \underset{\Delta}{\Rightarrow} g(g(a, f(a)), g(f(b), a))$$

with $\Delta = \{1.1, 2.2\}$.

Theorem 5.14 (Soundness Theorem) *Let $G \Rightarrow_{p,g} H$ be an evaluation step based on a rewrite rule $l \rightarrow r$. Then for each node v in G ,*

$$term_G(v) \underset{\Delta}{\Rightarrow} term_H(\text{track}(v))$$

through $l \rightarrow r$ and σ_g , where $\Delta = \text{node}_{G/v}^{-1}(g_V(\text{root}_{\diamond l}))$.

Proof Let the evaluation step be given by the following double pushout:

$$\begin{array}{ccccc} \underline{L} & \longleftarrow & \underline{K} & \xrightarrow{b} & \underline{R} \\ g \downarrow & & d \downarrow & & h \downarrow \\ G & \longleftarrow & D & \xrightarrow[c]{} & H \end{array}$$

The proposition is shown by a variant of bottom-up induction (which can be verified by induction on the number of nodes in a jungle). Let v be a node in G such that the proposition holds for each node v' with $v >_G v'$.

Case 1: $v = g_V(\text{root}_{\Delta l})$. Let o be an open node in \underline{L} . Then $v >_G g_V(o)$ because $\text{root}_{\Delta l} >_{\underline{L}} o$. So $\text{term}_G(g_V(o)) \rightrightarrows_{\Delta} \text{term}_H(\text{track}(g_V(o)))$ by induction hypothesis, where $\Delta = \emptyset$ since v is not in $G/g_V(o)$. That is, each open node o in \underline{L} satisfies

$$\text{term}_G(g_V(o)) = \text{term}_H(\text{track}(g_V(o))). \quad (5.1)$$

Now consider some variable x in r . x occurs also in l and hence there is a unique open node o_x in \underline{L} such that $\text{term}_L(o_x) = x$. Then $\text{track}(g_V(o_x)) = c_V(d_V(o_x)) = h_V(b_V(o_x))$; with (5.1) follows $\text{term}_G(g_V(o_x)) = \text{term}_H(h_V(b_V(o_x)))$. As $b_V(o_x)$ represents x (Lemma 5.11), the definition of induced substitutions in 4.9 gives

$$\sigma_g(x) = \sigma_h(x) \text{ for each variable } x \text{ in } r. \quad (5.2)$$

Thus

$$\begin{aligned} \text{term}_G(g_V(\text{root}_{\Delta l})) &= \sigma_g(\text{term}_L(\text{root}_{\Delta l})) && \text{Lemma 4.10} \\ &= \sigma_g(l) \\ &\xrightarrow{\{\lambda\}} \sigma_g(r) \\ &= \sigma_h(r) && (5.2) \\ &= \sigma_h(\text{term}_R(b_V(\text{root}_{\Delta l}))) && \text{Lemma 5.11} \\ &= \text{term}_H(h_V(b_V(\text{root}_{\Delta l}))) && \text{Lemma 4.10} \\ &= \text{term}_H(c_V(d_V(\text{root}_{\Delta l}))) \\ &= \text{term}_H(\text{track}(g_V(\text{root}_{\Delta l}))), \end{aligned}$$

which establishes the desired rewrite step since $\text{node}_{G/v}$ maps only λ to v .

Case 2: $v \neq g_V(\text{root}_{\Delta l})$. Then $e \in E_D$ for the edge e with $s_G(e) = v$. Let $t_G(e) = v_1 \dots v_n$ ($n \geq 0$). By induction hypothesis,

$$\text{term}_G(v_i) \rightrightarrows_{\Gamma_i} \text{term}_H(\text{track}(v_i))$$

with $\Gamma_i = \{\rho \in \text{Pos}(\text{term}_G(v_i)) \mid \text{node}_{G/v_i}(\rho) = g_V(\text{root}_{\Delta l})\}$ for $i = 1, \dots, n$. The position set Δ does not contain λ since $\text{node}_{G/v}(\lambda) = v \neq g_V(\text{root}_{\Delta l})$. So Δ can be written as follows:

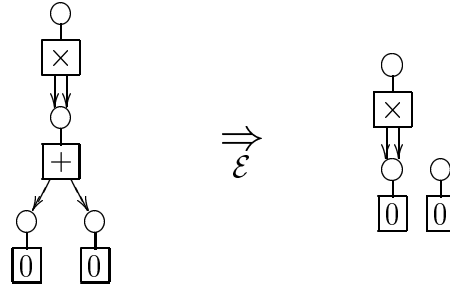
$$\begin{aligned} \Delta &= \{\pi \mid \text{node}_{G/v}(\pi) = g_V(\text{root}_{\Delta l})\} \\ &= \{i.\rho \mid 1 \leq i \leq n \text{ and } \text{node}_{G/v}(i.\rho) = g_V(\text{root}_{\Delta l})\} \\ &= \{i.\rho \mid 1 \leq i \leq n \text{ and } \text{node}_{G/v_i}(\rho) = g_V(\text{root}_{\Delta l})\} \\ &= \bigcup_{i=1}^n \{i.\rho \mid \rho \in \Gamma_i\}. \end{aligned}$$

Therefore

$$\begin{aligned}
term_G(v) &= m_G(e) term_G(v_1) \dots term_G(v_n) \\
&\Rightarrow_{\Delta} m_G(e) term_H(track(v_1)) \dots term_H(track(v_n)) \\
&= m_G(e) term_H^*(track^*(t_G(e))) \\
&= m_H(c_E(e)) term_H^*(c_V^*(t_D(e))) \\
&= m_H(c_E(e)) term_H^*(t_H(c_E(e))) \\
&= term_H(s_H(c_E(e))) \\
&= term_H(c_V(s_D(e))) \\
&= term_H(track(s_G(e))) \\
&= term_H(track(v)).
\end{aligned}$$

□

Example 5.15 The following evaluation step $G \Rightarrow_{\mathcal{E}} H$ is based on the second rewrite rule of Example 5.5:



Here $term_G(v) = (0 + 0) \times (0 + 0) \Rightarrow_{\Delta} 0 \times 0 = term_H(track(v))$ with $\Delta = \{1, 2\}$, where v is the topmost node in G .

For each rewrite step $term_G(v) \Rightarrow_{\Delta} term_H(track(v))$ induced by an evaluation step $G \Rightarrow_{\mathcal{E}} H$, the number of parallel replacements turns out to be the number of paths from v to $g_V(root_{\diamond l})$ in G (using the notations of Theorem 5.14). Before this result can be stated, the notion of a path has to be defined.

Definition 5.16 (path) Given two nodes v and v' in a jungle G , a *path* from v to v' is the empty sequence λ if $v = v'$ or else a non-empty sequence $\langle e_1, i_1 \rangle, \dots, \langle e_n, i_n \rangle$ such that $e_1, \dots, e_n \in E_G$, $v = s_G(e_1)$, $v' = t_G(e_n)|_{i_n}$, and $t_G(e_j)|_{i_j} = s_G(e_{j+1})$ for $j = 1, \dots, n - 1$.

Lemma 5.17 *For arbitrary nodes v and v' in a closed jungle G , the number of paths from v to v' is $|\text{node}_{G/v}^{-1}(v')|$.*

Proof By bottom-up induction on v . Let e be the edge with source v and assume that the proposition holds for each target node of e . If $v = v'$, then the proposition holds for v since $\text{node}_{G/v}$ maps only λ to v' . Assume therefore $v \neq v'$, and let $t_G(e) = v_1 \dots v_n$ ($n \geq 0$) and $\Delta = \text{node}_{G/v}^{-1}(v')$. Then $\Delta = \{i.\rho \mid 1 \leq i \leq n \text{ and } \text{node}_{G/v_i}(\rho) = v'\}$ and hence $|\Delta| = \sum_{i=1}^n |\text{node}_{G/v_i}^{-1}(v')|$. Thus, by induction hypothesis, $|\Delta|$ is the sum of the numbers of paths from v_i to v' , for $1 \leq i \leq n$. Clearly, this is just the number of paths from v to v' . \square

Given terms t, u , write $t \rightarrow_{\mathcal{R}}^0 u$ if $t = u$; for $n \geq 1$, write $t \rightarrow_{\mathcal{R}}^n u$ if there is t' such that $t \rightarrow_{\mathcal{R}} t' \rightarrow_{\mathcal{R}}^{n-1} u$ (so $\rightarrow_{\mathcal{R}}^n$ is the n -fold composition of $\rightarrow_{\mathcal{R}}$).

Corollary 5.18 *Let $G \Rightarrow_{p,g} H$ be an evaluation step based on a rewrite rule $l \rightarrow r$. Then for each node v in G ,*

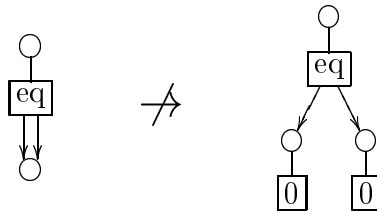
$$\text{term}_G(v) \xrightarrow{\mathcal{R}}^n \text{term}_H(\text{track}(v)),$$

where n is the number of paths from v to $g_V(\text{root}_{\Delta l})$.

Proof By combining the Soundness Theorem with Lemma 5.17 and the fact that for all terms t and u , $t \Rightarrow_{\Delta} u$ implies $t \rightarrow_{\mathcal{R}}^{|\Delta|} u$. \square

5.4 Applicability and Normal Forms

The results of the previous section show that every jungle evaluation step corresponds to a (parallel) application of the underlying term rewrite rule. In contrast, a rewrite rule $l \rightarrow r$ may be applicable to a term represented by a jungle G although the evaluation rule for $l \rightarrow r$ is not applicable to G . As an example, consider the rewrite rule $\text{eq}(x, x) \rightarrow \text{true}$ and the following jungles:



The left jungle is the left-hand side of the evaluation rule for $\text{eq}(x, x) \rightarrow \text{true}$, but there is no morphism to the right jungle although the rewrite rule is applicable to $\text{eq}(0, 0)$. Nevertheless, it is clear that the evaluation rule becomes applicable as soon as the two 0-edges have been folded. This example indicates that a proper realization of function evaluation requires the application of both evaluation and folding rules.

Before the applicability of evaluation rules can be characterized, the compatibility of jungle morphisms with the assignment of nodes to positions has to be shown.

Lemma 5.19 *Let G and H be closed jungles and $g: \underline{G} \rightarrow H$ be a jungle morphism. Then for each node v in G and each position π in $\text{term}_G(v)$,*

$$g_V(\text{node}_{G/v}(\pi)) = \text{node}_{H/g_V(v)}(\pi).$$

Proof By bottom-up induction in G . Let $v = s_G(e)$ for some edge e such that the proposition holds for each node in $t_G(e)$. If $\pi = \lambda$, then the proposition holds obviously for v . If $\pi = i.\rho$ for some $i \geq 1$ and $\rho \in \text{Pos}(\text{term}_G(v)/i)$, then $t_G(e) = v_1 \dots v_n$ for some $n \geq i$. Hence

$$\begin{aligned} g_V(\text{node}_{G/v}(i.\rho)) &= g_V(\text{node}_{G/v_i}(\rho)) \\ &= \text{node}_{H/g_V(v_i)}(\rho) && \text{induction hypothesis} \\ &= \text{node}_{H/s_H(g_E(e))}(i.\rho) \\ &= \text{node}_{H/g_V(s_G(e))}(i.\rho) \\ &= \text{node}_{H/g_V(v)}(i.\rho). \end{aligned}$$

□

The next lemma relates the different kinds of matching required for jungles and terms.

Lemma 5.20 (morphisms and substitutions) *Let G be a closed jungle, v be a node in G , and l be a term. Then there is a jungle morphism $g: \underline{\diamond}l \rightarrow G$ with $g_V(\text{root}_{\diamond}l) = v$ if and only if*

- (1) *there is a substitution σ such that $\text{term}_G(v) = \sigma(l)$, and*
- (2) *for all $\pi, \rho \in \text{Pos}(l)$ with $l/\pi = l/\rho \in \Sigma_X$, $\text{node}_{G/v}(\pi) = \text{node}_{G/v}(\rho)$.*

Proof Let $g: \underline{\diamond}l \rightarrow G$ be a jungle morphism with $g_V(\text{root}_{\diamond}l) = v$. By Lemma 4.10, $\text{term}_G(v) = \text{term}_G(g_V(\text{root}_{\diamond}l)) = \sigma_g(\text{term}_{\diamond}l(\text{root}_{\diamond}l)) = \sigma_g(l)$ for the substitution σ_g induced by g . To show that (2) is satisfied, let $\pi, \rho \in \text{Pos}(l)$ with $l/\pi = l/\rho \in$

Σ_X . Then $node_{\diamond l / root_{\diamond l}}(\pi)$ and $node_{\diamond l / root_{\diamond l}}(\rho)$ represent the same variable and hence are equal because $\diamond l$ is variable-collapsed. With Lemma 5.19 follows

$$\begin{aligned}
node_{G/v}(\pi) &= node_{G/g_V(root_{\diamond l})}(\pi) \\
&= g_V(node_{\diamond l / root_{\diamond l}}(\pi)) \\
&= g_V(node_{\diamond l / root_{\diamond l}}(\rho)) \\
&= node_{G/g_V(root_{\diamond l})}(\rho) \\
&= node_{G/v}(\rho).
\end{aligned}$$

Conversely, let (1) and (2) be satisfied. Let $v' \in V_{\diamond l}$ and $\pi \in Pos(l)$ such that $node_{\diamond l / root_{\diamond l}}(\pi) = v'$. Define $g_V: V_{\diamond l} \rightarrow V_G$ by setting $g_V(v') = node_{G/v}(\pi)$ (note that $V_{\diamond l} = V_{\diamond l}$). To see that g_V is well-defined, consider any $\rho \in Pos(l)$ with $node_{\diamond l / root_{\diamond l}}(\rho) = v'$. If $indegree_{\diamond l}(v') = 0$, then $\rho = \lambda = \pi$. If $indegree_{\diamond l}(v') = 1$, then there is a unique path from $root_{\diamond l}$ to v' and $\rho = \pi$ follows by Lemma 5.17. If $indegree_{\diamond l}(v') > 1$, then v' represents a variable so that $l/\pi = l/\rho \in \Sigma_X$; then (2) yields $node_{G/v}(\pi) = node_{G/v}(\rho)$. Thus g_V is a mapping. Moreover, $g_V(root_{\diamond l}) = g_V(node_{\diamond l / root_{\diamond l}}(\lambda)) = node_{G/v}(\lambda) = v$.

By Lemma 5.10,

$$\begin{aligned}
term_G(g_V(v')) &= term_G(node_{G/v}(\pi)) \\
&= term_G(v)/\pi \\
&= \sigma(l)/\pi \\
&= \sigma(l/\pi) \\
&= \sigma(term_{\diamond l}(root_{\diamond l})/\pi) \\
&= \sigma(term_{\diamond l}(node_{\diamond l / root_{\diamond l}}(\pi))) \\
&= \sigma(term_{\diamond l}(v'))
\end{aligned}$$

for each node v' in $\diamond l$ and each $\pi \in Pos(\diamond l)$ with $node_{\diamond l / root_{\diamond l}}(\pi) = v'$. It follows that g_V is label preserving, and for each edge e in $\diamond l$, $g_V(s_{\diamond l}(e))$ has an outgoing edge e' with the same label. Hence the assignment $e \mapsto e'$ determines a mapping $g_E: E_{\diamond l} \rightarrow E_G$ that preserves sources and labels. It remains to show that $g_V^*(t_{\diamond l}(e)) = t_G(e')$. Let $t_{\diamond l}(e) = v_1 \dots v_n$ and $t_G(e') = v'_1 \dots v'_n$ ($n \geq 0$). The case $n = 0$ is clear, so let $n \geq 1$ and consider some v_k , $1 \leq k \leq n$. Then there is $\pi \in Pos(l)$ such that $node_{\diamond l / root_{\diamond l}}(\pi) = s_{\diamond l}(e)$ and $node_{\diamond l / root_{\diamond l}}(\pi.k) = v_k$. From

$$\begin{aligned}
node_{G/v}(\pi) &= g_V(node_{\diamond l / root_{\diamond l}}(\pi)) \\
&= g_V(s_{\diamond l}(e)) \\
&= s_G(e')
\end{aligned}$$

follows $node_{G/v}(\pi.k) = v'_k$, so $g_V(v_k) = g_V(node_{\diamond l / root_{\diamond l}}(\pi.k)) = node_{G/v}(\pi.k) = v'_k$. Thus $g_V^*(t_{\diamond l}(e)) = t_G(e')$, and $g = \langle g_V, g_E \rangle$ is a jungle morphism. \square

The application of an evaluation rule ($\underline{L} \leftarrow \underline{K} \rightarrow \underline{R}$) to a jungle G requires a jungle morphism $\underline{L} \rightarrow G$ that satisfies the gluing condition (see Section 3.2).

While Lemma 5.20 provides necessary and sufficient conditions for the existence of a morphism $\underline{L} \rightarrow G$, the structure of \underline{L} and \underline{K} ensures that each such morphism satisfies the gluing condition.

Theorem 5.21 (applicability) *Given a closed jungle G , there is an evaluation step $G \Rightarrow_{\varepsilon} H$ based on a rewrite rule $l \rightarrow r$ if and only if there is a node v in G such that*

- (1) $term_G(v) = \sigma(l)$ for some substitution σ , and
- (2) for all $\pi, \rho \in Pos(l)$ with $l/\pi = l/\rho \in \Sigma_X$, $node_{G/v}(\pi) = node_{G/v}(\rho)$.

Proof Let $p = (\underline{L} \leftarrow \underline{K} \rightarrow \underline{R})$ be the evaluation rule for $l \rightarrow r$. If there is an evaluation step $G \Rightarrow_{p,g} H$, then g is a jungle morphism from $\underline{L} = \underline{\diamond}l$ to G and hence $v = g_V(root_{\diamond}l)$ satisfies (1) and (2) by Lemma 5.20.

Conversely, Lemma 5.20 shows that there is a morphism $g: \underline{\diamond}l \rightarrow G$ if there is $v \in V_G$ such that (1) and (2) are satisfied. So it remains to show that the gluing condition is satisfied. The contact condition holds because $V_{\underline{L}} - V_{\underline{K}}$ is empty. The identification condition holds also, since $E_{\underline{L}} - E_{\underline{K}} = \{e\}$, where e is the edge outgoing from $root_{\diamond}l$, and e cannot be identified by g with any other edge in $\underline{\diamond}l$. The latter follows from the fact that there is a path from $root_{\diamond}l$ to each other node v in $\underline{\diamond}l$ and hence $g_V(root_{\diamond}l) >_G g_V(v)$; then $g_V(root_{\diamond}l) \neq g_V(v)$ since G is acyclic. \square

Corollary 5.22 *Given a closed jungle G , there are jungles \tilde{G} and H such that $G \Rightarrow_{\mathcal{F}}^* \tilde{G} \Rightarrow_{\varepsilon} H$ if and only if a rewrite rule in \mathcal{R} is applicable to a term in $TERM_G$.*

Proof Assume that there is a derivation $G \Rightarrow_{\mathcal{F}}^* \tilde{G} \Rightarrow_{\varepsilon} H$. By Corollary 5.18, $term_{\tilde{G}}(v) \rightarrow_{\mathcal{R}} term_H(track(v))$ for the image v of the root in the left-hand side of the applied evaluation rule. Since $track_{G \Rightarrow_{\mathcal{F}}^* \tilde{G}}$ is surjective, there is $\bar{v} \in V_G$ with $track_{G \Rightarrow_{\mathcal{F}}^* \tilde{G}}(\bar{v}) = v$. Then $term_G(\bar{v}) \rightarrow_{\mathcal{R}} term_H(track(v))$ because folding preserves terms.

Conversely, let a rewrite rule $l \rightarrow r$ be applicable to $term_G(v)$, where v is some node in G . Then there is a substitution σ such that $\sigma(l)$ is a subterm of $term_G(v)$; so there is $v' \in V_G$ with $term_G(v') = \sigma(l)$ (Lemma 4.5). Let $G \Rightarrow_{\mathcal{F}}^* \tilde{G}$ be a folding such that \tilde{G} is fully collapsed. By Lemma 4.16, there is a jungle morphism $f: G \rightarrow \tilde{G}$ such that $term_{\tilde{G}} \circ f_V = term_G$. Hence for all positions π, ρ

in l with $l/\pi = l/\rho \in \Sigma_X$,

$$\begin{aligned}
term_{\tilde{G}}(node_{\tilde{G}/f_V(v')}(\pi)) &= term_{\tilde{G}}(f_V(node_{G/v'}(\pi))) && \text{Lemma 5.19} \\
&= term_G(node_{G/v'}(\pi)) \\
&= term_G(v')/\pi && \text{Lemma 5.10} \\
&= \sigma(l)/\pi \\
&= \sigma(l)/\rho \\
&\vdots \\
&= term_{\tilde{G}}(node_{\tilde{G}/f_V(v')}(\rho)).
\end{aligned}$$

Thus, by injectivity of $term_{\tilde{G}}$, $node_{\tilde{G}/f_V(v')}$ identifies π and ρ . According to Theorem 5.21, this implies the existence of an evaluation step $\tilde{G} \Rightarrow_{\mathcal{E}} H$. \square

The above proof uses a complete folding $G \Rightarrow_{\mathcal{F}}^* \tilde{G}$ to make the evaluation rule for $l \rightarrow r$ applicable. But it is often possible to do with less folding steps since Theorem 5.21 requires only that all occurrences of a variable in l correspond to the same node in G . In particular, G needs not be folded if l is a linear term, that is, if no variable occurs more than once in l .

As a consequence of Corollary 5.22, jungles represent term normal forms when neither evaluation nor folding rules can be applied.

Corollary 5.23 (normal forms) *A closed jungle G is a normal form with respect to $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ if and only if G is fully collapsed and all terms in $TERM_G$ are normal forms with respect to $\rightarrow_{\mathcal{R}}$.*

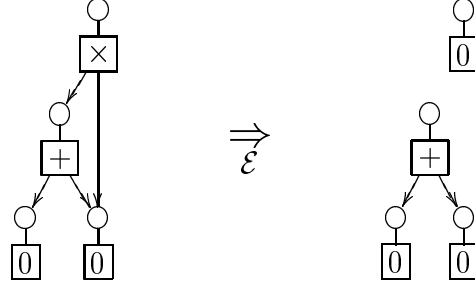
Proof If G is a normal form with respect to $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$, then G is fully collapsed by Lemma 4.17. Moreover, all terms in $TERM_G$ are normal forms as otherwise Corollary 5.22 would yield the contradiction that G is not a normal form.

Conversely, if G is fully collapsed, then no folding rule is applicable to G ; and if all terms in $TERM_G$ are normal forms, the Soundness Theorem implies that an evaluation step $G \Rightarrow_{\mathcal{E}} H$ is impossible. \square

5.5 Termination

In the light of the previous section, jungle evaluation has to comprise both evaluation and folding steps (the Completeness Theorem 6.13 together with Example 6.14 gives further evidence for the need of folding). The order in which the two kinds of steps are applied during evaluation should a priori not be restricted. So the rest of this chapter deals with the union $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ of both relations. In other words, jungle evaluation is identified with the abstract reduction system $\langle \mathfrak{J}, \Rightarrow_{\mathcal{E} \cup \mathcal{F}} \rangle$, where \mathfrak{J} is the set of isomorphism classes of closed jungles over Σ .

In this section jungle evaluation and term rewriting are compared with respect to their termination behaviour. Intuitively, termination of term rewriting should carry over to jungle evaluation since evaluation steps induce rewrite steps on represented terms. The problem for deriving a corresponding result is that evaluation steps may produce “garbage” that gives rise to additional rule applications. As an example, consider the evaluation step



which is based on the rewrite rule $x \times 0 \rightarrow 0$. The subjungle representing $0 + 0$ is preserved, while this subterm is removed in the rewrite step $(0+0) \times 0 \rightarrow_{\mathcal{R}} 0$ taking place at the topmost node v in the left jungle. The subjungle can be regarded as garbage in the right jungle because it does not contribute to the representation of the term at $track(v)$. Nevertheless, it gives rise to an evaluation step based on the rewrite rule $x + 0 \rightarrow x$ and to several folding steps.

The key to the proof that termination of $\rightarrow_{\mathcal{R}}$ carries over to $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is the multiset ordering proposed by Dershowitz and Manna [DM79]. A (finite) *multiset* over a set S is a function $\mathbf{M}: S \rightarrow \mathbb{N}$ with $\mathbf{M}(s) = 0$ almost everywhere. Intuitively, $\mathbf{M}(s)$ is the number of occurrences of s in the multiset. Union and difference of multisets are defined by $\mathbf{M} \cup \mathbf{N}(s) = \mathbf{M}(s) + \mathbf{N}(s)$ and $\mathbf{M} - \mathbf{N}(s) = \max(\mathbf{M}(s) - \mathbf{N}(s), 0)$, for all multisets \mathbf{M}, \mathbf{N} and each $s \in S$. Moreover, $\mathbf{M} \subseteq \mathbf{N}$ if $\mathbf{M}(s) \leq \mathbf{N}(s)$ for each $s \in S$, and $s \in \mathbf{M}$ means $\mathbf{M}(s) \geq 1$.

Let now \succ be a strict partial ordering on S , that is, an irreflexive and transitive binary relation. The ordering \gg on multisets over S is defined as follows: $\mathbf{M} \gg \mathbf{N}$ if there are multisets **Rem**, **Add** such that

- (1) $\emptyset \neq \mathbf{Rem} \subseteq \mathbf{M}$,
- (2) $\mathbf{N} = (\mathbf{M} - \mathbf{Rem}) \cup \mathbf{Add}$,
- (3) for each $a \in \mathbf{Add}$ there is $r \in \mathbf{Rem}$ with $r \succ a$.

Theorem 5.24 ([DM79]) *The relation \gg is terminating if and only if \succ is.*

Now the desired result can be proved by instantiating \succ with an appropriate ordering on $T(\Sigma)$ and by interpreting closed jungles as multisets of terms.

Theorem 5.25 (termination) *If $\rightarrow_{\mathcal{R}}$ is terminating, then so is $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$.*

Proof Let $\rightarrow_{\mathcal{R}}$ be terminating. Extend $\rightarrow_{\mathcal{R}}^+$ to a relation \succ by defining

$$t \succ u \text{ if } u \text{ is a subterm of some } t' \text{ with } t \rightarrow_{\mathcal{R}}^+ t'.$$

By embedding $\rightarrow_{\mathcal{R}}^+$ -steps into context, every sequence $t_1 \succ t_2 \succ t_3 \dots$ can be lifted to a sequence $\widehat{t}_1 \rightarrow_{\mathcal{R}}^+ \widehat{t}_2 \rightarrow_{\mathcal{R}}^+ \widehat{t}_3 \dots$ such that for $i \geq 1$, t_i is a subterm of \widehat{t}_i . It follows that \succ is transitive and terminating because $\rightarrow_{\mathcal{R}}^+$ is. In particular, \succ is a strict ordering. Assign to every closed jungle G a multiset \mathbf{G} on $T(\Sigma)$ by

$$\mathbf{G}(t) = |\{v \in V_G \mid \text{term}_G(v) = t\}|.$$

Theorem 5.24 states that the multiset extension \gg of \succ is terminating. Thus, to show that $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is terminating it suffices to show that for all closed jungles G and H ,

$$G \Rightarrow_{\mathcal{E}\cup\mathcal{F}} H \text{ implies } \mathbf{G} \gg \mathbf{H}.$$

If $G \Rightarrow_{\mathcal{F}} H$, then $\mathbf{H} = \mathbf{G} - \{t\}$, where t is the term represented by the two nodes in G that are identified, and hence $\mathbf{G} \gg \mathbf{H}$. Assume therefore $G \Rightarrow_{\mathcal{E}} H$, based on a rewrite rule $l \rightarrow r$. Let v be the image of $\text{root}_{\diamond l}$ in G . Then \mathbf{H} can be obtained from \mathbf{G} by removing all terms represented by nodes from which v is reachable, and by adding both the terms represented by the *track*-images of these nodes and the terms represented by the new nodes in H . More precisely, $\mathbf{H} = (\mathbf{G} - \text{Rem}) \cup \text{Add}$ with multisets **Rem** and **Add** defined as follows:

$$\text{Rem}(t) = |\{v' \in V_G \mid v' \geq_G v \text{ and } \text{term}_G(v') = t\}|$$

and $\text{Add} = \text{Old} \cup \text{New}$ with

$$\text{Old}(t) = \begin{cases} |\{v' \in V_G \mid v' >_G v \text{ and } \text{term}_H(\text{track}(v')) = t\}| & \text{if } r \in \Sigma_X, \\ |\{v' \in V_G \mid v' \geq_G v \text{ and } \text{term}_H(\text{track}(v')) = t\}| & \text{otherwise} \end{cases}$$

and

$$\text{New}(t) = |\{v' \in V_H \mid v' \notin \text{track}(V_G) \text{ and } \text{term}_H(v') = t\}|.$$

Rem is nonempty as it contains $\text{term}_G(v)$. Now consider some $t \in \text{Add}$.

Case 1: $t \in \text{Old}$. Then there is $v' \in V_G$ with $v' \geq_G v$ and $\text{term}_H(\text{track}(v')) = t$. Corollary 5.18 shows that $\text{term}_G(v') \rightarrow_{\mathcal{R}}^+ t$, so $\text{Rem} \ni \text{term}_G(v') \succ t$.

Case 2: $t \in \text{New}$. Then there is $v' \in V_H - \text{track}(V_G)$ with $\text{term}_H(v') = t$. The construction of evaluation rules (Definition 5.4) implies $\text{track}(v) >_H v'$, so t is a subterm of $\text{term}_H(\text{track}(v))$. Moreover, Corollary 5.18 yields $\text{term}_G(v) \rightarrow_{\mathcal{R}} \text{term}_H(\text{track}(v))$. Hence $\text{Rem} \ni \text{term}_G(v) \succ t$.

Thus $\mathbf{G} \gg \mathbf{H}$, showing that $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is terminating. \square

The following example demonstrates that the converse of Theorem 5.25 does not hold, that is, there are non-terminating term rewriting systems that become terminating under jungle evaluation.⁵

Example 5.26 Suppose that \mathcal{R} contains the following two rules ([Der87]):

$$\begin{array}{l} f(a, b, x) \rightarrow f(x, x, x) \\ b \rightarrow a \end{array}$$

Non-termination of $\rightarrow_{\mathcal{R}}$ is witnessed by the infinite sequence

$$f(a, b, b) \rightarrow_{\mathcal{R}} f(b, b, b) \rightarrow_{\mathcal{R}} f(a, b, b) \rightarrow_{\mathcal{R}} \dots$$

But $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is terminating since every step $G \Rightarrow_{\mathcal{E}\cup\mathcal{F}} H$ satisfies $\tau(G) > \tau(H) \geq 1$, where for a jungle X , $\tau(X) = n_X + |m_X^{-1}(b)| + |E_X|$ with n_X being the number of f -labelled edges the first two target nodes of which are distinct.

As another example, let \mathcal{R} be the system

$$\begin{array}{l} f(x) \rightarrow g(x, x) \\ a \rightarrow b \\ g(a, b) \rightarrow f(a) \end{array}$$

which is non-terminating due to the sequence

$$f(a) \rightarrow_{\mathcal{R}} g(a, a) \rightarrow_{\mathcal{R}} g(a, b) \rightarrow_{\mathcal{R}} f(a) \rightarrow_{\mathcal{R}} \dots$$

To see that $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is terminating, observe that no evaluation or folding step increases the number of g -labelled edges with distinct target nodes. Therefore an infinite sequence $G_1 \Rightarrow_{\mathcal{E}\cup\mathcal{F}} G_2 \Rightarrow_{\mathcal{E}\cup\mathcal{F}} \dots$ had to contain a jungle G_k such that the number of these edges remains constant in all G_n with $n \geq k$. So the sequence $G_k \Rightarrow_{\mathcal{E}\cup\mathcal{F}} G_{k+1} \Rightarrow_{\mathcal{E}\cup\mathcal{F}} \dots$ could not contain evaluation steps with the third rewrite rule. But an infinite sequence without these steps is impossible (assign to a jungle X the sum $|m_X^{-1}(f)| + |m_X^{-1}(a)| + |E_X|$ which decreases in each step).

The situation for normalization turns out to be opposite to the one for termination: normalization of jungle evaluation carries over to term rewriting, but not vice versa.

Theorem 5.27 (normalization) *If $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is normalizing, then so is $\rightarrow_{\mathcal{R}}$.*

Proof Given an arbitrary term t , choose a closed jungle G such that $term_G(v) = t$ for some node v in G . If $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is normalizing, then there is a derivation $G \Rightarrow_{\mathcal{E}\cup\mathcal{F}}^* H$ such that H is a normal form. Corollaries 5.18 and 5.23 imply that $term_H(track(v))$ is a normal form of t . \square

A counterexample that refutes the converse of Theorem 5.27 is given in the framework of collapsed tree rewriting, see Example 6.20.

⁵Two further examples of this kind can be found in Chapter 8. In those examples, non-termination arises from the union of two terminating systems.

5.6 Confluence

Unlike termination, confluence does not carry over from term rewriting to jungle evaluation. There are essentially two obstacles. Firstly, sharing common subexpressions prevents an independent evaluation of certain equal subterms. Secondly, jungle derivations may fail to be confluent on account of the garbage produced by evaluation steps. The first effect turns out to be harmless as long as jungle evaluation is normalizing. The second phenomenon, however, cannot be avoided without strong restrictions and hence suggests to abstract from the garbage contained in jungles.

Example 5.28 Let \mathcal{R} contain the single rule $a \rightarrow f(a)$ and suppose that Σ contains a binary function symbol g . Figure 5.4 shows two derivations outgoing from a jungle representing $g(a, a)$ such that the resulting jungles cannot derive a common jungle. So $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is non-confluent although $\rightarrow_{\mathcal{R}}$ is clearly confluent.

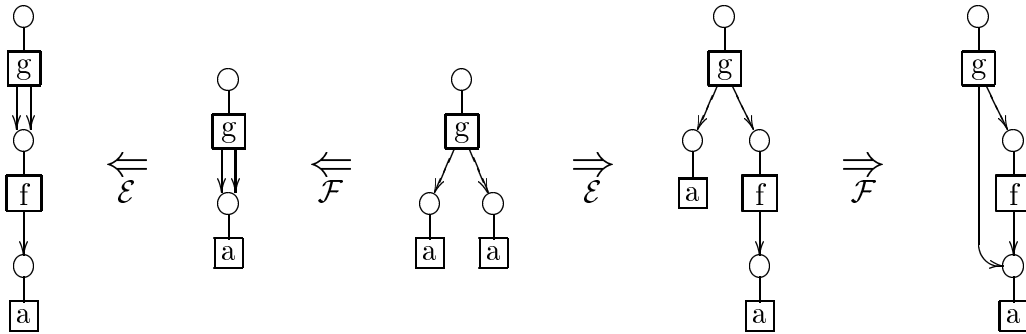


Figure 5.4: Non-confluence caused by sharing

In this example the sharing responsible for non-confluence is created by folding steps, but in Chapter 6 it is demonstrated that such a sharing can also result from evaluation steps alone (see Example 6.20). The next example addresses the failure of confluence due to garbage.

Example 5.29 The term rewriting system

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

is easily shown to be confluent. But Figure 5.5 demonstrates that the jungle representing the term a derives two normal forms that are non-isomorphic because of the garbage produced by the evaluation step based on $f(c) \rightarrow b$.

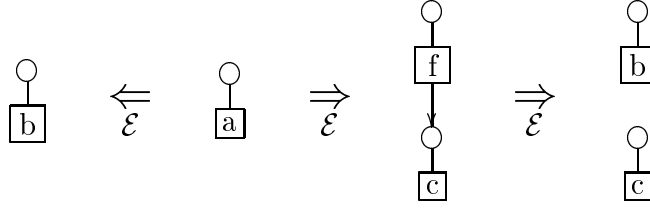


Figure 5.5: Non-confluence caused by garbage

To make precise what is meant by “garbage”, jungle nodes without ingoing edges are distinguished from other nodes.

Definition 5.30 (roots) The set of *roots* in a jungle G is given by $ROOT_G = \{v \in V_G \mid indegree_G(v) = 0\}$.

Now consider a jungle derivation $G \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* H$. Assuming that one is interested to evaluate the terms in $term_G(ROOT_G)$, all nodes in G are relevant in the sense that they contribute to the representation of these terms. To read off evaluated terms in H , all those nodes are relevant that are reachable from some node in $track(ROOT_G)$. All other nodes together with their outgoing edges can be considered as garbage with respect to the given derivation. The following definition modifies the confluence property by disregarding garbage.

Definition 5.31 (confluence up to garbage) The relation $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is *confluent up to garbage* if for all closed jungles G, H_1 , and H_2 with $H_1 \Leftarrow_{\mathcal{E} \cup \mathcal{F}}^* G \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* H_2$ there are jungles X_1 and X_2 such that for $i = 1, 2$, $H_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_i$ and $X_1^\bullet \cong X_2^\bullet$, where X_i^\bullet is the subjungle of X_i induced by

$$\{v \mid track_{G \Rightarrow^* H_i \Rightarrow^* X_i}(\bar{v}) \geq_{X_i} v \text{ for some } \bar{v} \in ROOT_G\}.$$

Theorem 5.32 (confluence) *If $\rightarrow_{\mathcal{R}}$ is confluent and $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ normalizing, then $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is confluent up to garbage.*

Proof Consider two jungle derivations $H_1 \Leftarrow_{\mathcal{E} \cup \mathcal{F}}^* G \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* H_2$. If $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is normalizing, then there is a derivation $H_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_i$ such that X_i is a normal form, for $i = 1, 2$. Let X_1^\bullet, X_2^\bullet be defined as in Definition 5.31. The plan is to show $TERM_{X_1^\bullet} = TERM_{X_2^\bullet}$. Let v be a node in X_1^\bullet . Then there are $\bar{v} \in ROOT_G$ and $v_1 \in X_1$ such that $track_{G \Rightarrow^* H_1 \Rightarrow^* X_1}(\bar{v}) = v_1 \geq_{X_1} v$. Let $v_2 = track_{G \Rightarrow^* H_2 \Rightarrow^* X_2}(\bar{v})$. By Corollaries 5.18 and 5.23, $term_{X_1}(v_1)$ and $term_{X_2}(v_2)$ are normal forms of $term_G(\bar{v})$. So confluence of $\rightarrow_{\mathcal{R}}$ implies $term_{X_1}(v_1) = term_{X_2}(v_2)$. It follows that $term_{X_1}(v)$ is a subterm of $term_{X_2}(v_2)$, and hence there is a node v' in X_2 such that $v_2 \geq_{X_2} v'$ and $term_{X_2}(v') = term_{X_1}(v)$. Thus $TERM_{X_1^\bullet} \subseteq TERM_{X_2^\bullet}$. The converse is shown analogously. Now Theorem 4.13 yields $X_1^\bullet \cong X_2^\bullet$, as X_1^\bullet and X_2^\bullet are fully collapsed by Lemma 4.17. \square

Example 5.33 Consider again the rewriting system

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

for which $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is shown to be terminating in Example 5.26. By structural induction for terms it can be shown that every term has a unique normal form, implying that $\rightarrow_{\mathcal{R}}$ is confluent. With Theorem 5.32 follows that $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is confluent up to garbage.

In the above theorem, normalization of $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ cannot be relaxed to normalization of $\rightarrow_{\mathcal{R}}$. This follows from Example 6.20 (which is presented in the setting of collapsed tree rewriting). On the other hand, by the previous section it is clear that termination of $\rightarrow_{\mathcal{R}}$ implies normalization of $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$.

Corollary 5.34 *If $\rightarrow_{\mathcal{R}}$ is terminating and confluent, then $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is terminating and confluent up to garbage.*

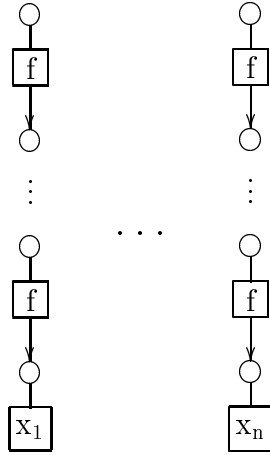
Proof Combine Theorems 5.25 and 5.32. □

In Example 5.33, $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is not only confluent up to garbage but also confluent. This can be shown by the “critical pair lemma” for hypergraph rewriting established in [Plu93b]. The next example demonstrates that confluence needs not imply confluence up to garbage, so the two properties are incomparable. Also, it turns out that confluence of $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ does not carry over to $\rightarrow_{\mathcal{R}}$ in general (this is again caused by garbage and is different for collapsed tree rewriting, see Theorem 6.19).

Example 5.35 Let \mathcal{R} contain the rules

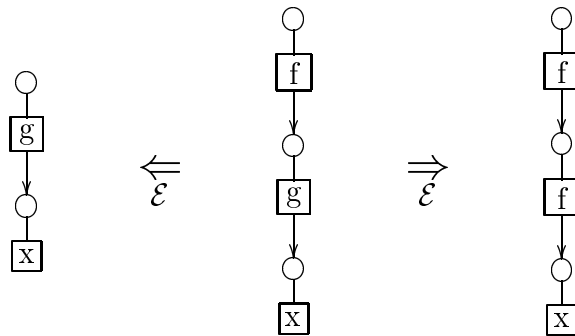
$$\begin{aligned} f(g(x)) &\rightarrow x \\ g(x) &\rightarrow f(x) \\ f(x) &\rightarrow f(f(x)) \end{aligned}$$

and suppose that f and g are the only function symbols in Σ . To see that $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is confluent, consider two derivations $H_1 \Leftarrow_{\mathcal{E} \cup \mathcal{F}}^* G \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* H_2$. Both H_1 and H_2 derive the following jungle by (1) applying the evaluation rule for $g(x) \rightarrow f(x)$ as long as possible, (2) applying folding rules as long as possible, and (3) performing evaluation steps based on $f(x) \rightarrow f(f(x))$:



Here x_1, \dots, x_n are the variables occurring in G , and each chain of f -labelled edges has the length of the longest chain of f - and g -edges in H_1 and H_2 .

Thus $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is confluent. But confluence up to garbage fails due to the following steps:



Denoting the left and right jungle by X_1 and X_2 , X_1^\bullet contains only the x -labelled edge and its source node whereas $X_2^\bullet = X_2$. It is evident that X_1^\bullet remains unchanged under rule applications to X_1 and that X_2^\bullet cannot reduce to X_1^\bullet . So $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is not confluent up to garbage. Moreover, $\rightarrow_{\mathcal{R}}$ is non-confluent as the terms resulting from the rewrite steps $x \leftarrow_{\mathcal{R}} f(g(x)) \rightarrow_{\mathcal{R}} f(f(x))$ have no common reduct.

Confluence properties are further discussed in Section 6.5 and Chapter 7, in the setting of collapsed tree rewriting. In particular, in Chapter 7 a criterion for (local) confluence based on “critical overlaps” of evaluation rules is developed.

Chapter 6

Collapsed Tree Rewriting

Section 5.6 reveals that it is necessary to ignore the garbage produced by evaluation steps in order to infer confluence of jungle evaluation from confluence of term rewriting. Garbage is also harmful with respect to the completeness behaviour of jungle evaluation. As a very simple example, let $f(a) \rightarrow b$ be the only rule in \mathcal{R} . Then $f(a)$ and b are equal in all models of \mathcal{R} , which is reflected by the equivalence $f(a) \leftrightarrow_{\mathcal{R}}^* b$ (cf. Theorem 5.2). But garbage prevents that the (minimal) jungles representing $f(a)$ and b are equivalent under $\Leftrightarrow_{\mathcal{E} \cup \mathcal{F}}^*$, see Figure 6.1. To establish

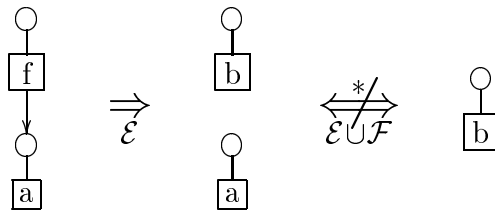


Figure 6.1: Incompleteness caused by garbage

completeness with respect to equational validity would require to compare jungles only up to garbage, in a similar fashion as in Section 5.6. In addition to these drawbacks, garbage gives rise to evaluation and folding steps that are useless for function evaluation and slow down the evaluation process.

The latter problem could be overcome by imposing the restriction that evaluation and folding steps do not take place in the garbage parts of jungles. Still, this would not support an elegant treatment of confluence and completeness. As a consequence, in this chapter evaluation steps are extended by “built-in garbage collection” and the class of jungles considered for evaluation is restricted to *collapsed trees*, that is, closed jungles with a unique root.

One should be aware that the removal of garbage by evaluation steps does not force real implementations to collect garbage “eagerly”: it just means that

garbage is not subject of evaluation or folding steps and that garbage collection is a separable task to be treated on a lower level of abstraction.

Before introducing collapsed tree rewriting in Section 6.2, it is shown how jungle derivations can be restricted to subjungles respectively extended by context. Both operations are needed several times in the sequel.

6.1 Restriction and Extension of Derivations

The following lemma is essential for the translation of jungle evaluation into collapsed tree rewriting given in the next section. It allows to restrict evaluation and folding steps to suitable subjungles (which need not be closed). The lemma is an analogue of Kreowski's Clip Theorem for general graph rewriting [Kre77].

Lemma 6.1 (Restriction Lemma) *Let $G \Rightarrow_{p,g} H$ be an evaluation or folding step through a rule $p = (L \leftarrow K \rightarrow R)$. Then for each subjungle U of G with $gL \subseteq U$ there is a direct derivation $U \Rightarrow_{p,g'} W$ such that*

- (1) g' is the restriction of g to U ,
- (2) $W \subseteq H$, and
- (3) $track_{U \Rightarrow W}$ is the restriction of $track_{G \Rightarrow H}$ to V_U and V_W .

Proof Let $G \Rightarrow_{p,g} H$ have the following diagram:

$$\begin{array}{ccccc}
 L & \longleftarrow & K & \xrightarrow{b} & R \\
 g \downarrow & [1] & d \downarrow & [2] & \downarrow h \\
 G & \longleftarrow & D & \xrightarrow{c} & H
 \end{array}$$

If $gL \subseteq U$, then g can be restricted to a jungle morphism $g': L \rightarrow U$. As g satisfies the gluing condition, so does g' . Hence, by Lemma 3.5, the square [1'] in the diagram below is a pushout, where D' is the subjungle of U with nodes and edges $U - (g'L - g'K)$, and d' is the restriction of g' to D' . By construction, D' is a subjungle of $D = G - (gL - gK)$ and d' is the restriction of d to D' .

$$\begin{array}{ccccc}
 L & \longleftarrow & K & \xrightarrow{b} & R \\
 g' \downarrow & [1'] & d' \downarrow & [2'] & \downarrow h' \\
 U & \longleftarrow & D' & \xrightarrow{c'} & H'
 \end{array}$$

Let now $[2']$ be pushout of b and d' , and $incl$ be the inclusion $D' \rightarrow D$. Then $h \circ b = c \circ d = c \circ incl \circ d'$ by commutativity of $[2]$, so the universal property of $[2']$ yields a jungle morphism $f: H' \rightarrow H$ satisfying $f \circ h' = h$ and $f \circ c' = c \circ incl$. One obtains the following commutative diagram:

$$\begin{array}{ccc}
K & \xrightarrow{b} & R \\
d' \downarrow & [2'] & \downarrow h' \\
D' & \xrightarrow{c'} & H' \\
incl \downarrow & [2''] & \downarrow f \\
D & \xrightarrow{c} & H
\end{array}$$

Since $[2']$ and the outer rectangle $[2'] \cup [2''] = [2]$ are pushouts, $[2'']$ is a pushout by Lemma 3.7(2). Moreover, by Lemma 3.6(4), f is injective and hence fH' is isomorphic to H' . Therefore $[2']$ remains a pushout after replacing H' by $W = fH'$, c' by $i \circ c'$, and h' by $i \circ h'$, where $i: H' \rightarrow W$ is the restriction of f . Combining the resulting pushout with $[1']$ yields a direct derivation $U \Rightarrow_{p,g'} W$. Moreover, $track_{U \Rightarrow W}$ is the restriction of $track_{G \Rightarrow H}$ because $track_{U \Rightarrow W}(v) = i(c'(v)) = f(c'(v)) = c(v) = track_{G \Rightarrow H}(v)$ for each node v in U . \square

The following Extension Lemma is an analogue of results established by Ehrig [Ehr77] and Kreowski [Kre77] for general graph rewriting. It shows that jungle derivations over evaluation and folding rules can be extended by arbitrary context. This is exploited, in particular, in Section 7.2 to extend critical pairs and their joining derivations.

Lemma 6.2 (Extension Lemma) *Let $U \Rightarrow_{\varepsilon \cup \mathcal{F}} U_1 \Rightarrow_{\varepsilon \cup \mathcal{F}} \dots \Rightarrow_{\varepsilon \cup \mathcal{F}} U_n$ be a derivation ($n \geq 1$) over possibly non-closed jungles and $g: U \rightarrow G$ be an injective jungle morphism. Then there is a derivation $G \Rightarrow_{\varepsilon \cup \mathcal{F}} G_1 \Rightarrow_{\varepsilon \cup \mathcal{F}} \dots \Rightarrow_{\varepsilon \cup \mathcal{F}} G_n$ such that G_n is defined by the pushout*

$$\begin{array}{ccc}
V & \xrightarrow{tr} & U_n \\
\bar{g} \downarrow & & \downarrow \\
J & \longrightarrow & G_n
\end{array}$$

where V is the discrete subjungle of U with node set V_U , J is the subjungle of G with node set V_G and edge set $E_G - g_E E_U$, \bar{g} is restriction of g , and tr is $track_{U \Rightarrow U_n}$ considered as a jungle morphism. Moreover, $track_{G \Rightarrow G_n}$ is the node mapping of $J \rightarrow G_n$.

Proof Let $U \Rightarrow_{\mathcal{E} \cup \mathcal{F}} U_1$ have the following diagram:

$$\begin{array}{ccccc}
 L & \longleftarrow & K & \longrightarrow & R \\
 f \downarrow & & [1] & & \downarrow \\
 & & \downarrow & & [2] \\
 & & & & \downarrow \\
 U & \longleftarrow & D & \xrightarrow{c} & U_1
 \end{array}$$

Let e be the unique edge in $E_U - E_D$ (i.e. $\{e\} = f_E E_L - f_E E_K$), F be the subjungle of G with node set V_G and edge set $E_G - \{g_E(e)\}$, and $g': D \rightarrow F$ be restriction of g . Construct square [4] in the diagram below as pushout of g' and c .

$$\begin{array}{ccccc}
 V & \longrightarrow & D & \xrightarrow{c} & U_1 \\
 \bar{g} \downarrow & & [3] & & \downarrow \\
 & & \downarrow g' & & [4] \\
 & & & & \downarrow g_1 \\
 J & \longrightarrow & F & \longrightarrow & G_1
 \end{array}$$

Moreover, let V , J , and \bar{g} be defined as in the proposition, and $V \rightarrow D$, $J \rightarrow F$ be inclusions. Then square [3] is commutative, that is, \bar{g} is restriction of g' . Since $E_J = E_G - g_E E_U = E_F - g_E E_D = E_F - g'_E E_D$ and $V_J = V_F = V_F - (g'_V V_D - g'_V V_V)$, [3] is a pushout by Lemma 3.5 (the gluing condition is satisfied because g' is injective and $V_D = V_V$). So, by Lemma 3.7(1), the outer rectangle [3] \cup [4] is a pushout. Also, letting $F \rightarrow G$ be inclusion, square [5] below is a pushout by Lemma 3.5, and hence a direct derivation $G \Rightarrow_{\mathcal{E} \cup \mathcal{F}} G_1$ is obtained by composing [1] with [5] and [2] with [4].

$$\begin{array}{ccc}
 U & \longleftarrow & D \\
 g \downarrow & & [5] \\
 & & \downarrow g' \\
 G & \longleftarrow & F
 \end{array}$$

Furthermore, the composition of $V \rightarrow D$ and c is just $track_{U \Rightarrow U_1}$ considered as a jungle morphism, and $track_{G \Rightarrow G_1}$ is the node mapping of $J \rightarrow F \rightarrow G_1$. Thus the proposition holds for the case $n = 1$.

Assume now $n > 1$ and, as induction hypothesis, that the proposition holds for all derivations containing less than n direct derivations. By Lemma 3.6(4), g_1 in pushout [4] is injective because \bar{g} is. Hence, by induction hypothesis, there is a derivation $G_1 \Rightarrow_{\mathcal{E} \cup \mathcal{F}} G_2 \Rightarrow_{\mathcal{E} \cup \mathcal{F}} \dots \Rightarrow_{\mathcal{E} \cup \mathcal{F}} G_n$ and a pushout

$$\begin{array}{ccc}
 V_1 & \xrightarrow{tr_1} & U_n \\
 \bar{g}_1 \downarrow & & [6] \\
 & & \downarrow \\
 J_1 & \longrightarrow & G_n
 \end{array}$$

the components of which are defined according to the proposition. Decompose the pushout [3] \cup [4] into the squares

$$\begin{array}{ccccc}
 V & \longrightarrow & V_1 & \longrightarrow & U_1 \\
 \bar{g} \downarrow & & [7] & \bar{g} \downarrow & [8] & \downarrow g_1 \\
 J & \longrightarrow & J_1 & \longrightarrow & G_1
 \end{array}$$

where $V_1 \rightarrow U_1$ and $J_1 \rightarrow G_1$ are inclusions, and $V \rightarrow V_1$ and $J \rightarrow J_1$ are restrictions of $V \rightarrow U_1$ respectively $J \rightarrow G_1$. The restriction $J \rightarrow J_1$ is well-defined since $E_{J_1} = E_{G_1} - g_{1E}E_{U_1}$ and, by Lemma 3.6(3), the image of J in G_1 has no common edges with g_1U_1 . Moreover, the node and edge sets of J_1 can be written as $G_1 - (g_1U_1 - g_1V_1)$ and hence [8] is a pushout by Lemma 3.5. With Lemma 3.7(3) follows that [7] is a pushout, because $V_1 \rightarrow U_1$ is injective. Thus, by composing squares [7] and [6] one obtains a pushout as stated in the proposition. Note that $V \rightarrow V_1$ is restriction of $V \rightarrow U_1$ which in turn is $track_{U \Rightarrow U_1}$ considered as a morphism. So $track_{U \Rightarrow U_1 \Rightarrow *U_n}$ considered as a morphism is just the composition of $V \rightarrow V_1$ and tr_1 . Finally, $track_{G \Rightarrow G_1 \Rightarrow *G_n}$ is the node mapping of $J \rightarrow J_1 \rightarrow G_n$ since (1) $track_{G_1 \Rightarrow *G_n}$ satisfies this property with respect to $J_1 \rightarrow G_n$ by induction hypothesis, and (2) $J \rightarrow J_1$ is restriction of $J \rightarrow G_1$ the node mapping of which is $track_{G \Rightarrow G_1}$ (as shown above). \square

6.2 Getting Rid of Garbage

In this section jungle evaluation is modified to collapsed tree rewriting by restricting attention to jungles having a single root. To preserve this structure, the removal of garbage is incorporated in evaluation steps. Then the new computational model is related to jungle evaluation by means of the Restriction Lemma presented in the last section.

Definition 6.3 (collapsed tree) A closed jungle C is a *collapsed tree*¹ if $ROOT_C$ contains a single node.

The unique root of a collapsed tree C is denoted by $root_C$ and $term(C)$ stands for $term_C(root_C)$. Observe that $root_C \geq_C v$ for each node v in C . Note also that for a closed jungle G and $v \in V_G$, the “collapsed tree at v ” G/v (as introduced in Definition 5.8) is a collapsed tree in the above sense. For a collapsed tree C , the mapping $node_{C/root_C} : Pos(term(C)) \rightarrow V_C$ is from now on denoted by $node_C$.

¹The term “collapsed tree” goes back (at least) to Walker and Strong [WS73]. Their collapsed trees, however, are always fully collapsed.

Lemma 6.4 (1) For each node v in a closed jungle G , $\text{term}_G(v) = \text{term}(G/v)$.

(2) Let $f: G \rightarrow H$ be a jungle morphism between closed jungles. Then for each node v in G , $f(G/v) = H/f_V(v)$.

Proof (1) Using Lemma 4.6, $\text{term}_G(v) = \text{term}_G(\text{incl}(v)) = \text{term}_{G/v}(v) = \text{term}(G/v)$, where incl is the inclusion of G/v in G .

(2) Top-down induction shows that a node v' in H belongs to $f(G/v)$ if and only if $f_V(v) \geq_H v'$. Then $f(G/v)$ and $H/f_V(v)$ contain the same nodes and edges, and hence are equal. \square

Definition 6.5 ($\Rightarrow_{\mathcal{E}}, \Rightarrow_{\mathcal{R}}$) The relations $\Rightarrow_{\mathcal{E}}$ and $\Rightarrow_{\mathcal{R}}$ on collapsed trees are defined as follows:

(1) $C \Rightarrow_{\mathcal{E}} D$ if there is an evaluation step $C \Rightarrow_{\mathcal{E}} M$ with $M/\text{track}(\text{root}_C) = D$.

(2) $C \Rightarrow_{\mathcal{R}} D$ if $C \Rightarrow_{\mathcal{E}} D$ or $C \Rightarrow_{\mathcal{F}} D$.²

Steps over $\Rightarrow_{\mathcal{E}}$ are called evaluation steps, as in the case of jungle evaluation. Also, notations and terminology for hypergraph derivations of Section 3.3 are taken over for $\Rightarrow_{\mathcal{E}}$ and $\Rightarrow_{\mathcal{R}}$. In particular, the track function is adapted in the obvious way: $\text{track}_{C \Rightarrow_{\mathcal{E}} D}$ is undefined for all nodes that have descendants in $V_M - V_D$.

The results for jungle evaluation concerning soundness, applicability of rules, and normal forms carry over to collapsed tree rewriting in a straightforward way. As an example, soundness is established in the following theorem.

Theorem 6.6 (soundness) Let $C \Rightarrow_{p,g} D$ be an evaluation step based on a rewrite rule $l \rightarrow r$ in \mathcal{R} . Then

$$\text{term}(C) \xrightarrow[\mathcal{R}]^n \text{term}(D),$$

where n is the number of paths from root_C to $g_V(\text{root}_{\Delta l})$.

Proof Let $C \Rightarrow_{\mathcal{E}} M$ such that $M/\text{track}(\text{root}_C) = D$. By Corollary 5.18 and Lemma 6.4(1),

$$\begin{aligned} \text{term}(C) &= \text{term}_C(\text{root}_C) \\ &\xrightarrow[\mathcal{R}]^n \text{term}_M(\text{track}(\text{root}_C)) \\ &= \text{term}(M/\text{track}(\text{root}_C)) \\ &= \text{term}(D), \end{aligned}$$

with n as in the proposition. \square

²The notation “ $\Rightarrow_{\mathcal{R}}$ ” is given preference over “ $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ ” to indicate the strong relationship to the term rewrite relation $\rightarrow_{\mathcal{R}}$.

Note that the number n of rewrite steps from $\text{term}(C)$ to $\text{term}(D)$ is at least 1, since each node in C is reachable from root_C . That is, $C \Rightarrow_{\mathcal{E}} D$ implies $\text{term}(C) \rightarrow_{\mathcal{R}}^+ \text{term}(D)$.

In contrast to the corresponding proof for jungle evaluation, here soundness implies immediately that collapsed tree rewriting is terminating whenever term rewriting is.

Corollary 6.7 (termination) *If $\rightarrow_{\mathcal{R}}$ is terminating, then so is $\Rightarrow_{\mathcal{R}}$.*

Proof Every infinite sequence $C_1 \Rightarrow_{\mathcal{R}} C_2 \Rightarrow_{\mathcal{R}} \dots$ contains infinitely many $\Rightarrow_{\mathcal{E}}$ -steps because $\Rightarrow_{\mathcal{F}}$ is terminating. Also, folding steps preserve terms, so Theorem 6.6 yields an infinite sequence of $\rightarrow_{\mathcal{R}}^+$ -steps. Thus $\Rightarrow_{\mathcal{R}}$ is terminating if $\rightarrow_{\mathcal{R}}$ is. \square

Collapsed tree rewriting is henceforth identified with the abstract reduction system $\langle \mathfrak{C}, \Rightarrow_{\mathcal{R}} \rangle$, where \mathfrak{C} is the set of isomorphism classes of collapsed trees over Σ . In the subsequent sections, completeness of collapsed tree rewriting, termination in relation to jungle evaluation, and confluence properties are examined.

The next lemma provides a translation of jungle evaluation into collapsed tree rewriting in the following way: Given some closed jungle G and a step $G \Rightarrow_{\mathcal{E} \cup \mathcal{F}} H$, each collapsed tree at a node in G is either subject of an $\Rightarrow_{\mathcal{R}}$ -step or is preserved up to isomorphism. This translation is exploited, for instance, in Sections 7.2 and 8.2 to apply Theorem 3.8 (commutation of independent direct derivations) to collapsed tree rewriting.

Lemma 6.8 (Translation Lemma) *Let G and H be closed jungles such that $G \Rightarrow_{\mathcal{E} \cup \mathcal{F}} H$. Then for each node v in G , $G/v \Rightarrow_{\mathcal{R}}^{\lambda} H/\text{track}_{G \Rightarrow H}(v)$.*

Proof Let $G \Rightarrow_{p,g} H$ for some evaluation or folding rule $p = (L \leftarrow K \rightarrow R)$, and consider an arbitrary node v in G .

Case 1: $gL \subseteq G/v$. Then, by Lemma 6.1, there is a step $G/v \Rightarrow_{p,g'} W \subseteq H$ the track function of which is the restriction of $\text{track}_{G \Rightarrow H}$. Hence, by Definition 6.5 and Lemma 6.4(2),

$$\begin{aligned} G/v &\Rightarrow_{\mathcal{R}} W/\text{track}_{G/v \Rightarrow W}(\text{root}_{G/v}) \\ &= W/\text{track}_{G \Rightarrow H}(v) \\ &= \text{incl}(W/\text{track}_{G \Rightarrow H}(v)) \\ &= H/\text{track}_{G \Rightarrow H}(v), \end{aligned}$$

where incl is the inclusion of W in H .

Case 2: $gL \not\subseteq G/v$. Let $G \Rightarrow_{p,g} H$ have the following diagram:

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ g \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \xrightarrow{c} & H \end{array}$$

Case 2.1: p is an evaluation rule. Then the unique edge e removed from G is not in G/v , as otherwise $gL \subseteq G/v$. Hence $G/v \subseteq D$. The construction of evaluation rules in Definition 5.4 ensures that if c identifies two nodes, then $s_D(e)$ is one of these nodes. It follows that c is injective on G/v . Thus $G/v \cong c(G/v) = H/c_V(v) = H/track_{G \Rightarrow H}(v)$.

Case 2.2: p is a folding rule. By Lemma 4.16(2) there is a jungle morphism $f: G \rightarrow H$ with $f_V = track_{G \Rightarrow H}$. Moreover, folding rules are defined such that $track_{G \Rightarrow H}$ identifies only the source nodes of the two edges in gL . By assumption, at least one of these nodes is not in G/v . Consequently f is injective on G/v , and hence $G/v \cong f(G/v) = H/f_V(v) = H/track_{G \Rightarrow H}(v)$. \square

Corollary 6.9 *Let G and H be closed jungles such that $G \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* H$. Then for each node v in G , $G/v \Rightarrow_{\mathcal{R}}^* H/track_{G \Rightarrow^* H}(v)$.*

Proof By induction on the length of $G \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* H$, using Lemma 6.8. \square

6.3 Completeness

Collapsed tree rewriting turns out to be complete with respect to equational validity in the same sense as term rewriting is. More precisely, the Completeness Theorem established below shows that for arbitrary collapsed trees C and D ,

$$C \underset{\mathcal{R}}{\overset{*}{\Leftrightarrow}} D \text{ if and only if } Mod(\mathcal{R}) \models term(C) \doteq term(D).$$

So collapsed tree rewriting is a complete mechanism for proving equations over specifications that are presented as term rewriting systems. An important aspect of this result is that it does *not* hold when folding is discarded from the rewrite relation $\Rightarrow_{\mathcal{R}}$ (this is demonstrated in Example 6.14). As a consequence, graph rewriting approaches that perform expression evaluation solely by evaluation steps lack this kind of completeness. (This applies, for example, to the approaches of Barendregt et al. [BvEG⁺87] and Corradini and Rossi [CR93a].)

By Theorem 5.2, completeness can be equivalently expressed as “ $C \underset{\mathcal{R}}{\overset{*}{\Leftrightarrow}} D$ if and only if $term(C) \leftrightarrow_{\mathcal{R}}^* term(D)$,” and this property is shown in the proof of the Completeness Theorem. While soundness of collapsed tree rewriting rests on the fact that sequences of $\Rightarrow_{\mathcal{R}}$ -steps induce sequences of $\rightarrow_{\mathcal{R}}$ -steps, there is no such direct correspondence from which completeness could be obtained. As an example, consider the rewrite rules $f(x) \rightarrow g(x, x)$ and $a \rightarrow b$. Then $f(a) \rightarrow_{\mathcal{R}} g(a, a) \rightarrow_{\mathcal{R}} g(a, b)$, but there is no corresponding sequence of $\Rightarrow_{\mathcal{R}}$ -steps because of the sharing introduced by evaluation steps based on the first rule. Collapsed tree rewriting is nevertheless complete: the point is that $\underset{\mathcal{R}}{\overset{*}{\Leftrightarrow}}$ comprises “reversed folding steps” which allow to unfold collapsed trees between evaluation steps. Since every single term rewrite step $t \rightarrow_{\mathcal{R}} u$ can be simulated by

choosing for t a collapsed tree that is appropriately folded, unfolding can be used to lift $t \rightarrow_{\mathcal{R}} u$ to an $\Leftrightarrow_{\mathcal{R}}^*$ -conversion of the trees representing t and u . This is stated in the *Lifting Lemma*, leading straightforward to the completeness result.

At first two distinguished representations of terms are introduced, namely those with a minimal respectively maximal degree of sharing.

Definition 6.10 (tree and completely folded tree) A collapsed tree is a *tree* if no node has an indegree greater than one. A collapsed tree is a *completely folded tree*³ if it is fully collapsed.

For every term t , let Δt and $\blacktriangle t$ denote a tree respectively a completely folded tree representing t . It is easy to show that for every tree T with $\text{term}(T) = t$, the assignment $\text{root}_{\Delta t} \mapsto \text{root}_T$ extends to an isomorphism $\Delta t \rightarrow T$. So Δt is determined uniquely up to isomorphism. The uniqueness of $\blacktriangle t$ follows from Theorem 4.13 and Lemma 4.5, the latter implying $\text{TERM}_{\Delta t} = \text{TERM}_C$ for each collapsed tree C representing t .

The next lemma shows that Δt can be folded to each collapsed tree representing t , which in turn can be folded to $\blacktriangle t$.

Lemma 6.11 (existence of foldings) *For every collapsed tree C ,*

$$\Delta \text{term}(C) \xrightarrow[\mathcal{F}]{*} C \xrightarrow[\mathcal{F}]{*} \blacktriangle \text{term}(C).$$

Proof By Lemma 4.17 there is a folding $C \Rightarrow_{\mathcal{F}}^* \tilde{C}$ such that \tilde{C} is fully collapsed. Then $\tilde{C} \cong \blacktriangle \text{term}(C)$ since folding steps transform collapsed trees into collapsed trees.

To show the first part of the proposition, assign to every collapsed tree D the product $\phi_D = \prod_{v \in V} \text{indegree}_D(v)$ with $V = V_D - \{\text{root}_D\}$.

Claim. For every collapsed tree C , either C is a tree or there is a collapsed tree D such that $C \Leftarrow_{\mathcal{F}} D$ and $\phi_C > \phi_D$.

Proof. The application of a folding rule to a collapsed tree produces a node with an indegree greater than one, because each of the two identified nodes has an indegree of at least one. So folding steps cannot result in trees.

If C is not a tree, then there is a node v with $\text{indegree}_C(v) > 1$. Let e be the edge with $s_C(e) = v$. Define a collapsed tree D by adding to C a node v' having the label of v and an edge e' having the label of e . Define $s_D(e') = v'$ and $t_D(e') = t_D(e)$. Moreover, choose some edge a in C such that $v \in t_C(a)$, and obtain $t_D(a)$ by replacing some occurrence of v in $t_C(a)$ with v' . Then D is a collapsed tree such that $D \Rightarrow_{\mathcal{F}} C$ by applying the folding rule for $m_D(e)$ to e and e' . Furthermore, $\phi_D = \text{indegree}_D(v) \times \text{indegree}_D(v') \times \prod_{u \in U} \text{indegree}_C(u)$ with

³to avoid the phrase “fully collapsed collapsed tree”

$U = V_C - \{\text{root}_C, v\}$. It follows $\phi_C > \phi_D$ since $\text{indegree}_D(v) = \text{indegree}_C(v) - 1$ and $\text{indegree}_D(v') = 1$. \square

This claim implies that for every collapsed tree C there is a sequence

$$C = C_0 \leftarrow_{\mathcal{F}} C_1 \leftarrow_{\mathcal{F}} \dots \leftarrow_{\mathcal{F}} C_n$$

($n \geq 0$) such that C_n is a tree; this is because $\phi_D \geq 1$ for every collapsed tree D . Then $C_n \cong \Delta \text{term}(C)$ by the uniqueness of trees. \square

In the proof of the following Lifting Lemma, a term rewrite step $t \rightarrow_{\mathcal{R}} u$ is simulated by representing t as the collapsed tree C that is obtained from Δt by folding completely the subtree representing the replaced subterm in t . Rewriting C through the evaluation rule for the rule applied in $t \rightarrow_{\mathcal{R}} u$ yields then a collapsed tree D that can be unfolded to Δu . As an example, consider the rewrite step $g(f(0 + 0), 0) \rightarrow_{\mathcal{R}} g(f(0) + f(0), 0)$ through the rule $f(x + x) \rightarrow f(x) + f(x)$. The result of lifting this step to a conversion of trees is shown in Figure 6.2.

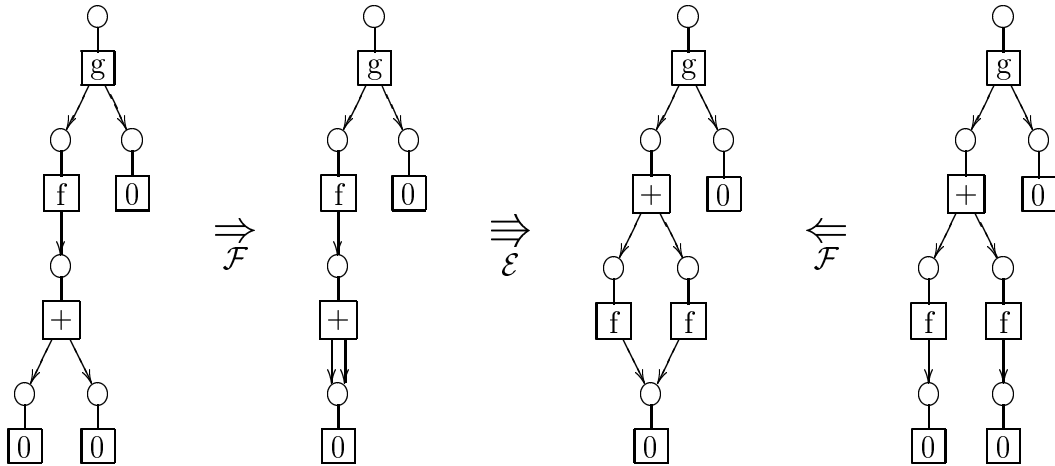


Figure 6.2: A lifted term rewrite step

Lemma 6.12 (Lifting Lemma) *For every term rewrite step $t \rightarrow_{\mathcal{R}} u$ there are collapsed trees C and D such that*

$$\Delta t \xrightarrow[\mathcal{F}]{} C \xrightarrow[\varepsilon]{} D \xrightarrow[\mathcal{F}]{} \Delta u.$$

Proof Let $l \rightarrow r$ be the rule applied in $t \rightarrow_{\mathcal{R}} u$, σ be the associated substitution, and π be the position in t where $\sigma(l)$ is replaced by $\sigma(r)$. Construct a folding $\Delta t = C_1 \Rightarrow_{\mathcal{F}} C_2 \Rightarrow_{\mathcal{F}} \dots \Rightarrow_{\mathcal{F}} C_n = C$ by repeating the step “apply

a folding rule in $C_i/\text{node}_{C_i}(\pi)$ ” as long as possible. Then $C/\text{node}_C(\pi)$ is fully collapsed. Also, by Lemma 5.10, $\text{term}_C(\text{node}_C(\pi)) = \text{term}(C)/\pi = t/\pi = \sigma(l)$. Let $v = \text{node}_C(\pi)$. By Lemma 5.20, there is a jungle morphism $g: \underline{\diamond}l \rightarrow C$ with $g_V(\text{root}_{\diamond}l) = v$ if $\text{node}_{C/v}$ identifies each two positions in l that represent the same variable. Let $\rho, \tau \in \text{Pos}(l)$ such that $l/\rho = l/\tau \in \Sigma_X$. With Lemma 5.10 follows $\text{term}_{C/v}(\text{node}_{C/v}(\rho)) = \text{term}_{C/v}(\text{node}_{C/v}(\tau))$, and hence $\text{node}_{C/v}(\rho) = \text{node}_{C/v}(\tau)$ by injectivity of $\text{term}_{C/v}$. Thus, by the proof of Theorem 5.21, there is an evaluation step $C \Rightarrow_{p,g} M$ with p being the evaluation rule for $l \rightarrow r$. So $C \Rightarrow_{\mathcal{E}} D$ for $D = M/\text{track}(\text{root}_C)$. The Soundness Theorem 5.14 shows that $\text{term}(C) \Rightarrow_{\Delta} \text{term}(D)$ through $l \rightarrow r$, where $\Delta = \text{node}_C^{-1}(v)$. By construction of the folding $\Delta t \Rightarrow_{\mathcal{F}}^* C$, there is only one path from root_C to v ; so Δ contains a single position (Lemma 5.17). Therefore $t = \text{term}(C) \Rightarrow_{\{\pi\}} \text{term}(D)$ through $l \rightarrow r$, which implies $\text{term}(D) = u$. Then, finally, $\Delta u \Rightarrow_{\mathcal{F}}^* D$ by Lemma 6.11. \square

Note that beside the translation of term rewrite steps into conversions of trees, there is another canonical way of lifting which results in conversions of completely folded trees. Namely, every step $t \rightarrow_{\mathcal{R}} u$ gives rise to a conversion $\blacktriangle t \Leftarrow_{\mathcal{F}}^* C \Rightarrow_{\mathcal{E}} D \Rightarrow_{\mathcal{F}}^* \blacktriangle u$, where C and D are chosen as in the above proof.

Theorem 6.13 (Completeness Theorem) *For all collapsed trees C and D ,*

$$C \Leftrightarrow_{\mathcal{R}}^* D \text{ if and only if } \text{term}(C) \leftrightarrow_{\mathcal{R}}^* \text{term}(D).$$

Proof Assume $C \Leftrightarrow_{\mathcal{R}}^* D$. If $C \cong D$, then clearly $\text{term}(C) = \text{term}(D)$. Otherwise there is a sequence $C = C_0 \Leftrightarrow_{\mathcal{R}} C_1 \Leftrightarrow_{\mathcal{R}} \dots \Leftrightarrow_{\mathcal{R}} C_n = D$. By Theorem 6.6 and Lemma 4.16, $X \Leftrightarrow_{\mathcal{R}} Y$ implies $\text{term}(X) \leftrightarrow_{\mathcal{R}}^* \text{term}(Y)$, for all collapsed trees X, Y . So induction on n yields $\text{term}(C) \leftrightarrow_{\mathcal{R}}^* \text{term}(D)$.

Conversely, let $\text{term}(C) \leftrightarrow_{\mathcal{R}}^* \text{term}(D)$. Induction on the number of $\leftrightarrow_{\mathcal{R}}$ -steps constituting this equivalence proves $\Delta \text{term}(C) \Leftrightarrow_{\mathcal{R}}^* \Delta \text{term}(D)$, by use of the Lifting Lemma. Then $C \Leftarrow_{\mathcal{F}}^* \Delta \text{term}(C) \Leftrightarrow_{\mathcal{R}}^* \Delta \text{term}(D) \Rightarrow_{\mathcal{F}}^* D$ by Lemma 6.11. \square

The following example demonstrates that completeness gets lost when folding is relinquished, that is, when $\Leftrightarrow_{\mathcal{R}}^*$ is replaced by $\Leftrightarrow_{\mathcal{E}}^*$.

Example 6.14 Let \mathcal{R} contain the following rules:

$$\begin{aligned} g(x) &\rightarrow p(x, f(x)) \\ h(x) &\rightarrow p(f(x), x) \\ a &\rightarrow f(a) \end{aligned}$$

Then $g(a) \leftrightarrow_{\mathcal{R}}^* h(a)$ since both terms are rewritable to $p(f(a), f(a))$. A conversion from $\Delta g(a)$ into $\Delta h(a)$ by collapsed tree rewriting is shown in Figure 6.3. To see

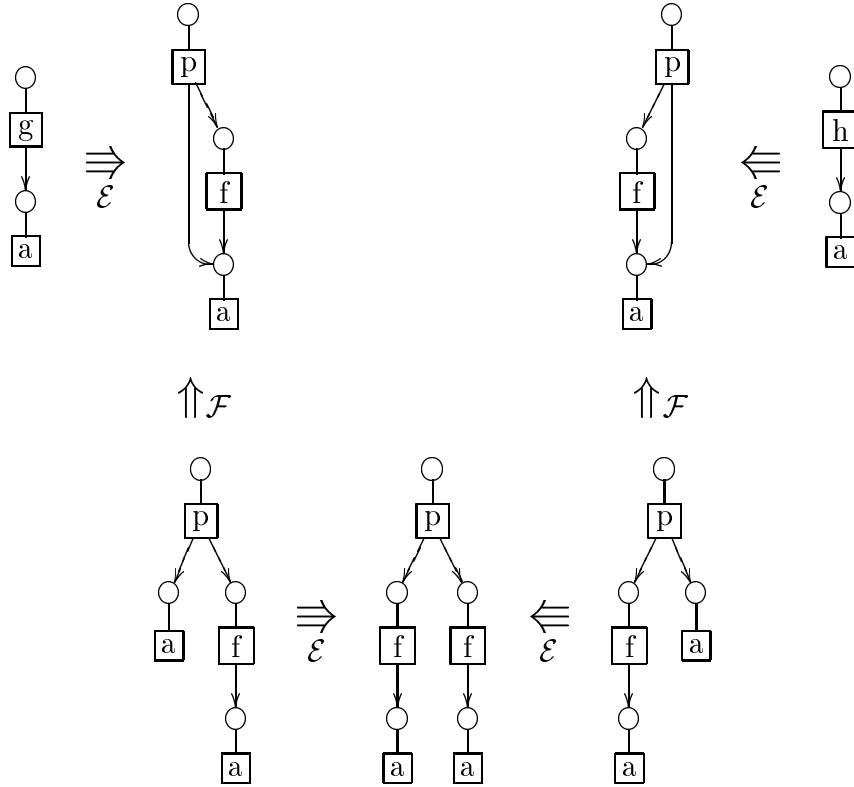


Figure 6.3: A conversion from $\Delta g(a)$ into $\Delta h(a)$

that folding steps are necessary for such a conversion, observe that every sequence $\Delta g(a) \Leftrightarrow_{\mathcal{R}} C_1 \Leftrightarrow_{\mathcal{R}} \dots \Leftrightarrow_{\mathcal{R}} C_n \Leftrightarrow_{\mathcal{R}} \Delta h(a)$ contains two evaluation steps that are based on the first two rules. Therefore such a sequence includes a conversion from $\blacktriangle p(t, f(t))$ into $\blacktriangle p(f(u), u)$, where t, u are of the form $f^n(a)$ with $n \geq 0$. But, obviously, this conversion must contain folding steps.

The Completeness Theorem shows that all equations valid in $Mod(\mathcal{R})$ have proofs by collapsed tree rewriting; but in general it is hopeless to search for such proofs since they involve reversed evaluation and folding steps which lead to a huge search space. As in the case of term rewriting, the situation is improved if one deals with confluent systems. Then the problem of finding a proof reduces to the search for common reducts of collapsed trees.

Corollary 6.15 *If $\Rightarrow_{\mathcal{R}}$ is confluent, then for all collapsed trees C and D ,*

$$term(C) \leftrightarrow_{\mathcal{R}}^* term(D) \text{ if and only if } C \Rightarrow_{\mathcal{R}}^* X \Leftarrow_{\mathcal{R}}^* D$$

for some collapsed tree X .

Proof By the equivalence of confluence and the Church-Rosser property (see Lemma 2.1(2)), $C \Leftrightarrow_{\mathcal{R}}^* D$ holds if and only if there is some collapsed tree X such that $C \Rightarrow_{\mathcal{R}}^* X \Leftarrow_{\mathcal{R}}^* D$. Hence, the proposition follows from the Completeness Theorem. \square

An immediate consequence of Corollary 6.15 is that the decision procedure for equational validity shown in Figure 6.4 is partially correct whenever $\Rightarrow_{\mathcal{R}}$ is confluent. For in this case two collapsed trees that possess normal forms have a common reduct if and only if their normal forms represent the same term (or, equivalently, are isomorphic). The procedure can be made totally correct

Given an equation $t \doteq u$, rewrite $\blacktriangle t$ and $\blacktriangle u$ to $\Rightarrow_{\mathcal{R}}$ -normal forms $NF_{\blacktriangle t}$ and $NF_{\blacktriangle u}$, and return the result of the test $term(NF_{\blacktriangle t}) =^{\Gamma} term(NF_{\blacktriangle u})$.

Figure 6.4: Decision procedure for equational validity

if there is an algorithm that computes a normal form for every collapsed tree. This is the case if $\Rightarrow_{\mathcal{R}}$ is normalizing and \mathcal{R} contains finitely many rules: then all derivations starting from a collapsed tree can be enumerated until a normal form is found. Finally, if $\Rightarrow_{\mathcal{R}}$ is not only normalizing but terminating, then a simple (nondeterministic) way of normalization is to perform $\Rightarrow_{\mathcal{R}}$ -steps as long as possible.

6.4 Termination

In this section it is shown that collapsed tree rewriting is terminating if and only if jungle evaluation is terminating. This equivalence is exploited in Chapter 7 to characterize confluence of terminating systems by means of critical pairs, and in Chapter 8 to prove a modularity result for termination.

The harder direction in the equivalence proof consists in showing that $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is terminating whenever $\Rightarrow_{\mathcal{R}}$ is. This is established similar to the proof of Theorem 5.25 in which the termination of $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is inferred from the termination of $\rightarrow_{\mathcal{R}}$. Actually, Theorem 5.25 turns out to be a corollary of the characterization proved here since, by Corollary 6.7, $\Rightarrow_{\mathcal{R}}$ is terminating whenever $\rightarrow_{\mathcal{R}}$ is.

The proof that termination of $\Rightarrow_{\mathcal{R}}$ carries over to $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ makes use of the following lemma which allows to extend $\Rightarrow_{\mathcal{R}}$ -steps by context.

Lemma 6.16 (extension of $\Rightarrow_{\mathcal{R}}$ -steps) *Given a rewrite step $A \Rightarrow_{\mathcal{R}} B$ and an injective jungle morphism $c: A \rightarrow C$ such that C is a collapsed tree, there is a step $C \Rightarrow_{\mathcal{R}} D$ and an injective morphism $d: B \rightarrow D$.*

Proof By Definition 6.5 there is a step $A \Rightarrow_{\mathcal{E}\cup\mathcal{F}} M$ with $M/\text{track}(\text{root}_A) = B$. So the Extension Lemma 6.2 yields a step $C \Rightarrow_{\mathcal{E}\cup\mathcal{F}} N$ such that N is defined by the pushout

$$\begin{array}{ccc} V & \xrightarrow{tr} & M \\ \bar{c} \downarrow & & \downarrow d' \\ J & \xrightarrow{f} & N \end{array}$$

where V is the discrete subjungle of A with node set V_A , J is the subjungle of C with node set V_C and edge set $E_C - c_E E_A$, \bar{c} is restriction of c , and tr is $\text{track}_{A \Rightarrow M}$ considered as a jungle morphism. Moreover, $f_V = \text{track}_{C \Rightarrow N}$. Since $\text{root}_C \geq_C c_V(\text{root}_A)$ and J is obtained from C by removing edges the source nodes of which are reachable from $c_V(\text{root}_A)$, $\text{root}_C \geq_J c_V(\text{root}_A)$ holds. Hence $f_V(\text{root}_C) \geq_N f_V(c_V(\text{root}_A))$. Let $D = N/\text{track}_{C \Rightarrow N}(\text{root}_C)$. Then $C \Rightarrow_{\mathcal{R}} D$ and

$$\begin{aligned} \text{root}_D &= \text{track}_{C \Rightarrow N}(\text{root}_C) \\ &= f_V(\text{root}_C) \\ &\geq_N f_V(c_V(\text{root}_A)) \\ &= f_V(\bar{c}_V(\text{root}_A)) \\ &= d'_V(tr(\text{root}_A)) \\ &= d'_V(\text{root}_B) \\ &= \text{root}_{d'B}. \end{aligned}$$

It follows $d'B \subseteq D$ and hence d' has a restriction $d: B \rightarrow D$. Finally, d and d' are injective because d' lies opposite to \bar{c} in the above pushout diagram. \square

Theorem 6.17 *The relation $\Rightarrow_{\mathcal{R}}$ is terminating if and only if $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is.*

Proof Assume at first that $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is terminating. By Definition 6.5, for every step $C \Rightarrow_{\mathcal{R}} D$ there is a direct derivation $C \Rightarrow_{\mathcal{E}\cup\mathcal{F}} M$ such that $M/\text{track}(\text{root}_C) = D$. Given an injective jungle morphism $C \rightarrow G$, the Extension Lemma 6.2 shows that there is a step $G \Rightarrow_{\mathcal{E}\cup\mathcal{F}} H$ and an injective morphism $M \rightarrow H$. Thus, since the composition of the inclusion $D \rightarrow M$ with $M \rightarrow H$ yields an injective morphism $D \rightarrow H$, every sequence $C_1 \Rightarrow_{\mathcal{R}} C_2 \Rightarrow_{\mathcal{R}} C_3 \Rightarrow_{\mathcal{R}} \dots$ gives rise to a sequence $G_1 \Rightarrow_{\mathcal{E}\cup\mathcal{F}} G_2 \Rightarrow_{\mathcal{E}\cup\mathcal{F}} G_3 \Rightarrow_{\mathcal{E}\cup\mathcal{F}} \dots$ with $G_1 = C_1$. It follows that $\Rightarrow_{\mathcal{R}}$ is terminating if $\Rightarrow_{\mathcal{E}\cup\mathcal{F}}$ is.

Conversely, let now $\Rightarrow_{\mathcal{R}}$ be terminating. Extend $\Rightarrow_{\mathcal{R}}$ to a relation \succ on collapsed trees⁴ by

$$C \succ F \text{ if } C \Rightarrow_{\mathcal{R}}^{\dagger} D \text{ for some } D \text{ such that } F \subseteq D.$$

⁴To be precise, \succ has to be considered as a relation on isomorphism classes of collapsed trees; see the next footnote.

To see that \succ is terminating, suppose there were an infinite sequence $C_1 \succ C_2 \succ C_3 \succ \dots$. Lemma 6.16 implies that whenever $A \Rightarrow_{\mathcal{R}}^+ B$ and there is an injective jungle morphism $A \rightarrow C$ into a collapsed tree C , then $C \Rightarrow_{\mathcal{R}}^+ D$ for some collapsed tree D such that there is an injective morphism $B \rightarrow D$. Hence, by definition of \succ , the sequence $C_1 \succ C_2 \succ C_3 \succ \dots$ can be extended to a sequence $D_1 \Rightarrow_{\mathcal{R}}^+ D_2 \Rightarrow_{\mathcal{R}}^+ D_3 \Rightarrow_{\mathcal{R}}^+ \dots$ with $D_1 = C_1$. But this contradicts the termination of $\Rightarrow_{\mathcal{R}}$. So \succ is terminating and, in particular, irreflexive. Moreover, with Lemma 6.16 it is easy to show that \succ is transitive.

Thus, by Theorem 5.24, the extension \gg of \succ to finite multisets of collapsed trees⁵ is also terminating (see page 46 for the definition of \gg). Now assign to every closed jungle G a multiset \mathbf{G} of collapsed trees by

$$\mathbf{G}(X) = |\{v \in V_G \mid G/v \cong X\}|.$$

It remains to show that for all closed jungles G and H ,

$$G \Rightarrow_{\mathcal{E} \cup \mathcal{F}} H \text{ implies } \mathbf{G} \gg \mathbf{H}.$$

Case 1: $G \Rightarrow_{\mathcal{F}} H$. Let v_1 and v_2 be the two nodes in G that are identified and \mathbf{Top} be the multiset defined by

$$\mathbf{Top}(X) = |\{v \in V_G \mid v_1, v_2 \in G/v \text{ and } G/v \cong X\}|.$$

Then $\mathbf{H} = (\mathbf{G} - \mathbf{Rem}) \cup \mathbf{Add}$ with $\mathbf{Rem} = \mathbf{Top} \cup \{G/v_1\}$ and \mathbf{Add} being defined by

$$\mathbf{Add}(X) = |\{v \in V_G \mid G/v \in \mathbf{Top} \text{ and } H/\text{track}(v) \cong X\}|.$$

\mathbf{Rem} is nonempty as it contains G/v_1 . Consider some $X \in \mathbf{Add}$ and $v \in V_G$ such that $G/v \in \mathbf{Top}$ and $H/\text{track}(v) \cong X$. Then $G/v \Rightarrow_{\mathcal{R}} X$ by the proof of the Translation Lemma 6.8 since the image of the left-hand side of the applied folding rule lies in G/v . Hence $\mathbf{Rem} \ni G/v \succ X$.

Case 2: $G \Rightarrow_{\mathcal{E}} H$. Let this step be based on a rewrite rule $l \rightarrow r$ and let v be the image of the root of the left-hand side of the applied evaluation rule. Then $\mathbf{H} = (\mathbf{G} - \mathbf{Rem}) \cup \mathbf{Add}$ with

$$\mathbf{Rem}(X) = |\{v' \in V_G \mid v' \geq_G v \text{ and } G/v' \cong X\}|$$

and $\mathbf{Add} = \mathbf{Old} \cup \mathbf{New}$, where

$$\mathbf{Old}(X) = \begin{cases} |\{v' \in V_G \mid v' >_G v \text{ and } H/\text{track}(v') \cong X\}| & \text{if } r \in \Sigma_X, \\ |\{v' \in V_G \mid v' \geq_G v \text{ and } H/\text{track}(v') \cong X\}| & \text{otherwise} \end{cases}$$

⁵The following multisets should be considered as multisets of isomorphism classes of collapsed trees in order to obey the finiteness condition.

and

$$\text{New}(X) = |\{v' \in V_H \mid v' \notin \text{track}(V_G) \text{ and } H/v' \cong X\}|.$$

It is clear that **Rem** is nonempty since $G/v \in \text{Rem}$. Now consider some $X \in \text{Add}$.

Case 2.1: $X \in \text{Old}$. Then $X \cong H/\text{track}(v')$ for some $v' \in V_G$ with $v' \geq_G v$. So the image of the left-hand side of the applied evaluation rule lies in G/v' and hence $G/v' \Rightarrow_{\mathcal{R}} X$ by the proof of the Translation Lemma 6.8. Hence $\text{Rem} \ni G/v' \succ X$.

Case 2.2: $X \in \text{New}$. Then $X \cong H/v'$ for some $v' \in V_H$ with $v' \notin \text{track}(V_G)$. The construction of evaluation rules assures $\text{track}(v) >_H v'$, so $H/v' \subseteq H/\text{track}(v)$. The proof of the Translation Lemma 6.8 shows that $G/v \Rightarrow_{\mathcal{R}} H/\text{track}(v)$, and hence $G/v \Rightarrow_{\mathcal{R}} C$ for a collapsed tree C such that $C \cong H/\text{track}(v)$ and $X \subseteq C$. Thus $\text{Rem} \ni G/v \succ X$.

Cases 1 and 2 prove $G \gg H$, establishing the termination of $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$. \square

6.5 Confluence and Unique Normal Forms

In this section the discussion of confluence properties of Section 5.6 is continued, working now in the setting of collapsed tree rewriting. By Example 5.28 it is already clear that confluence does not carry over from term rewriting to collapsed tree rewriting: $\Rightarrow_{\mathcal{R}}$ is non-confluent if \mathcal{R} contains the single rule $a \rightarrow f(a)$ and if there is a binary function symbol in Σ (having as argument sorts the sort of the constant a). One may object to this example that $\rightarrow_{\mathcal{R}}$ is not normalizing and that non-confluence is caused by the interaction of evaluation and folding steps. But below a counterexample is presented showing that even if $\rightarrow_{\mathcal{R}}$ is confluent and normalizing, neither $\Rightarrow_{\mathcal{E}}$ nor $\Rightarrow_{\mathcal{R}}$ needs be locally confluent.

Despite this relationship, it turns out that collapsed tree rewriting has unique normal forms if and only if term rewriting has. As a consequence, normalizing collapsed tree rewriting is confluent whenever term rewriting is, and convergence of term rewriting carries over to collapsed tree rewriting.

In contrast to the non-transferability of (local) confluence from term rewriting to collapsed tree rewriting, it turns out that $\rightarrow_{\mathcal{R}}$ is confluent respectively locally confluent whenever $\Rightarrow_{\mathcal{R}}$ is. So (local) confluence behaves with respect to transferability opposite to termination. The following lemma is useful for establishing this relation.

Lemma 6.18 *For every term rewrite step $t \rightarrow_{\mathcal{R}} u$ there is a collapsed tree C such that $\blacktriangle t \Rightarrow_{\mathcal{R}} C$ and $u \rightarrow_{\mathcal{R}}^* \text{term}(C)$. Pictorially:*

$$\begin{array}{ccc} t & \xrightarrow{\mathcal{R}} & u \xrightarrow{\mathcal{R}}^* \text{term}(C) \\ \uparrow & & \nearrow \\ \blacktriangle t & \xRightarrow{\mathcal{R}} & C \end{array}$$

Proof Let $l \rightarrow r$ be the rule applied in $t \rightarrow_{\mathcal{R}} u$, σ be the associated substitution, and π be the position in t where $\sigma(l)$ is replaced by $\sigma(r)$. By Lemma 5.10, $\text{term}_{\blacktriangle t}(\text{node}_{\blacktriangle t}(\pi)) = \text{term}(\blacktriangle t)/\pi = \sigma(l)$. Let $v = \text{node}_{\blacktriangle t}(\pi)$. Since $\blacktriangle t$ is fully collapsed, $\text{node}_{\blacktriangle t/v}$ identifies each two positions in l that represent the same variable. Hence, by Lemma 5.20 and the proof of Theorem 5.21, there is an evaluation step $\blacktriangle t \Rightarrow_{p,g} M$ based on $l \rightarrow r$ and such that $g_V(\text{root}_{\triangleleft l}) = v$. So $\blacktriangle t \Rightarrow_{\varepsilon} C$ for $C = M/\text{track}(\text{root}_{\blacktriangle t})$ and, by the Soundness Theorem 5.14, $t = \text{term}(\blacktriangle t) \Rightarrow_{\Delta} \text{term}(C)$ through $l \rightarrow r$, where $\Delta = \text{node}_{\blacktriangle t}^{-1}(v)$. As π is in Δ , it follows $t \Rightarrow_{\{\pi\}} u \Rightarrow_{\Delta-\{\pi\}} \text{term}(C)$. Then in particular $u \rightarrow_{\mathcal{R}}^* \text{term}(C)$. \square

Theorem 6.19 *If $\Rightarrow_{\mathcal{R}}$ is (locally) confluent, then so is $\rightarrow_{\mathcal{R}}$.*

Proof Assume that $\Rightarrow_{\mathcal{R}}$ is locally confluent. Given any two steps $s \leftarrow_{\mathcal{R}} t \rightarrow_{\mathcal{R}} u$, Lemma 6.18 yields two steps $C \leftarrow_{\mathcal{R}} \blacktriangle t \Rightarrow_{\mathcal{R}} D$ such that $s \rightarrow_{\mathcal{R}}^* \text{term}(C)$ and $u \rightarrow_{\mathcal{R}}^* \text{term}(D)$. By local confluence of $\Rightarrow_{\mathcal{R}}$, there is a collapsed tree X such that $C \Rightarrow_{\mathcal{R}}^* X \Leftarrow_{\mathcal{R}}^* D$. Theorem 6.6 implies $\text{term}(C) \rightarrow_{\mathcal{R}}^* \text{term}(X) \leftarrow_{\mathcal{R}}^* \text{term}(D)$, and hence $s \rightarrow_{\mathcal{R}}^* \text{term}(X) \leftarrow_{\mathcal{R}}^* u$. So $\rightarrow_{\mathcal{R}}$ is locally confluent.

Assume now that $\Rightarrow_{\mathcal{R}}$ is confluent. Let $s \leftarrow_{\mathcal{R}}^* t \rightarrow_{\mathcal{R}}^* u$ for some terms s, t, u , and consider collapsed trees C, D with $\text{term}(C) = s$ and $\text{term}(D) = u$. Then $s \leftarrow_{\mathcal{R}}^* u$ and hence, by Corollary 6.15, there is a collapsed tree X such that $C \Rightarrow_{\mathcal{R}}^* X \Leftarrow_{\mathcal{R}}^* D$. With Theorem 6.6 follows $s \rightarrow_{\mathcal{R}}^* \text{term}(X) \leftarrow_{\mathcal{R}}^* u$. Thus $\rightarrow_{\mathcal{R}}$ is confluent. \square

The following example demonstrates that the converse of Theorem 6.19 does not hold, even if $\rightarrow_{\mathcal{R}}$ is normalizing.

Example 6.20 Suppose that \mathcal{R} is given as follows⁶:

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

Based on the following rewrite steps, induction on the term structure shows that every term has a unique normal form:

$$\begin{array}{ccccc} c & \xleftarrow{\mathcal{R}} & g(a, b) & \xrightarrow{\mathcal{R}} & g(b, b) & \xrightarrow{\mathcal{R}} & f(a) \\ & & & & & & \downarrow \mathcal{R} \\ & & & & & & g(a, a) \\ & & & & \swarrow \mathcal{R} & & \\ & & & & & & \end{array}$$

So $\rightarrow_{\mathcal{R}}$ is normalizing and confluent. But Figure 6.5 shows that neither $\Rightarrow_{\mathcal{R}}$ nor $\Rightarrow_{\varepsilon}$ is locally confluent. The reason is that the sharing of the constant a in $\blacktriangle g(a, a)$ prevents a rewrite step from $\blacktriangle g(a, a)$ to $\Delta g(a, b)$.

⁶Middeldorp and Hamoen independently invented this system as a counterexample to the completeness of basic narrowing in the presence of confluence and normalization, see [MH92].

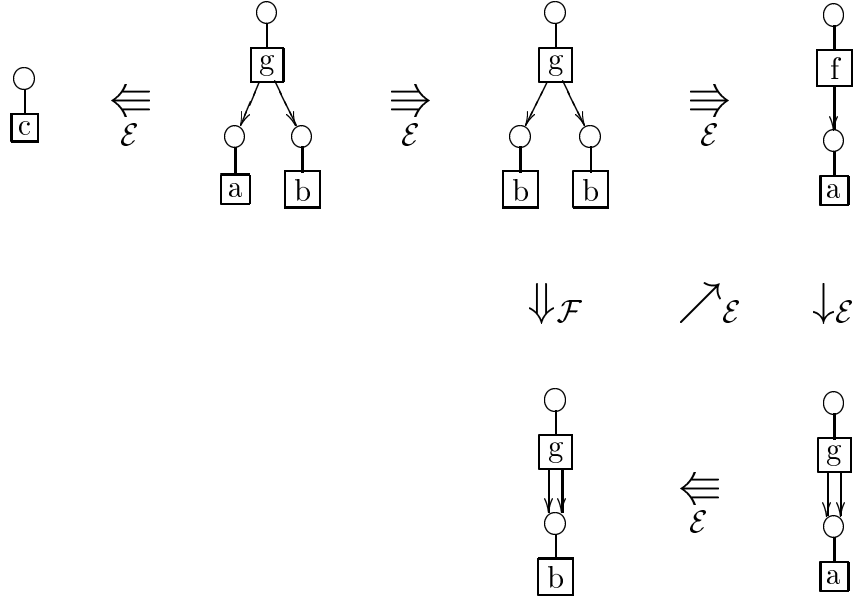


Figure 6.5: Non-confluence of $\Rightarrow_{\mathcal{R}}$ and $\Rightarrow_{\mathcal{E}}$

Although collapsed tree rewriting needs not be confluent when term rewriting is, the former has unique normal forms if and only if the latter has. Recall that the relation \rightarrow of an abstract reduction system $\langle A, \rightarrow \rangle$ has *unique normal forms* if for all normal forms a and b in A , $a \leftrightarrow^* b$ implies $a = b$.

Theorem 6.21 *The relation $\Rightarrow_{\mathcal{R}}$ has unique normal forms if and only if $\rightarrow_{\mathcal{R}}$ has.*

Proof Let $\Rightarrow_{\mathcal{R}}$ have unique normal forms, and consider term normal forms t and t' such that $t \leftrightarrow_{\mathcal{R}}^* t'$. Then $\blacktriangle t \Leftrightarrow_{\mathcal{R}}^* \blacktriangle t'$ by the Completeness Theorem, and Corollary 5.23 shows that $\blacktriangle t$ and $\blacktriangle t'$ are $\Rightarrow_{\mathcal{R}}$ -normal forms (note that a collapsed tree is a normal form w.r.t. $\Rightarrow_{\mathcal{R}}$ if and only if it is a normal form w.r.t. $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$). So $\blacktriangle t \cong \blacktriangle t'$ by the uniqueness of normal forms, implying $t = t'$.

Conversely, assume that $\rightarrow_{\mathcal{R}}$ has unique normal forms, and let C and C' be $\Rightarrow_{\mathcal{R}}$ -normal forms such that $C \Leftrightarrow_{\mathcal{R}}^* C'$. Then $\text{term}(C) \leftrightarrow_{\mathcal{R}}^* \text{term}(C')$ by soundness, and both terms are normal forms by Corollary 5.23. Uniqueness of normal forms gives $\text{term}(C) = \text{term}(C')$, and hence $C \cong C'$ by Theorem 4.13 since C and C' are fully collapsed. \square

Corollary 6.22 *Suppose that $\Rightarrow_{\mathcal{R}}$ is normalizing. Then $\Rightarrow_{\mathcal{R}}$ is confluent if and only if $\rightarrow_{\mathcal{R}}$ is.*

Proof By Theorem 6.19, $\rightarrow_{\mathcal{R}}$ is confluent whenever $\Rightarrow_{\mathcal{R}}$ is. Conversely, if $\rightarrow_{\mathcal{R}}$ is confluent, then it has in unique normal forms (Lemma 2.1(5)). With Theorem 6.21 follows that $\Rightarrow_{\mathcal{R}}$ has also unique normal forms. But it is easy to show that a normalizing relation with unique normal forms is confluent. \square

Note that the normalization of $\Rightarrow_{\mathcal{R}}$ cannot be weakened to normalization of $\rightarrow_{\mathcal{R}}$, in view of Example 6.20.

The proof that $\Rightarrow_{\mathcal{R}}$ is confluent in Corollary 6.22 (via Theorem 6.21) corresponds to Curien's and Ghelli's method for proving confluence in normalizing abstract reduction systems [CG91]. They show uniqueness of normal forms by an interpretation function from the system under consideration into a system that is known to be confluent. The function must be compatible with derivability in the two systems, preserve normal forms, and be injective on normal forms. In the present context these conditions are satisfied by the function *term* which interprets collapsed trees as terms (to be precise, the extension of *term* to isomorphism classes of collapsed trees is as required).

As an immediate consequence of Corollary 6.22, convergence carries over from term rewriting to collapsed tree rewriting. This corresponds to Corollary 5.34 which states that jungle evaluation is terminating and confluent up to garbage if term rewriting is convergent.

Corollary 6.23 *If $\rightarrow_{\mathcal{R}}$ is convergent, then so is $\Rightarrow_{\mathcal{R}}$.*

Proof Combine Corollaries 6.7 and 6.22. \square

The converse of this result does not hold, since $\rightarrow_{\mathcal{R}}$ needs not be terminating when $\Rightarrow_{\mathcal{R}}$ is convergent. A counterexample⁷ is again provided by the system

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

of Example 5.33, for which $\rightarrow_{\mathcal{R}}$ is confluent and non-terminating while $\Rightarrow_{\mathcal{R}}$ is terminating. By Corollary 6.22, $\Rightarrow_{\mathcal{R}}$ is confluent and hence convergent.

So, analogously to the situation for termination, convergent term rewriting systems form a proper subclass of the class of systems that are convergent under collapsed tree rewriting. As a result, the decision procedure for equational validity in Figure 6.4, which reduces collapsed trees nondeterministically to normal forms and checks equality of the resulting terms, terminates for more systems than the corresponding procedure by term rewriting. For instance, in deciding the validity of $f(a) \doteq g(b, b)$ over the above system, term rewriting may run into the loop $f(a) \rightarrow_{\mathcal{R}} g(a, a) \rightarrow_{\mathcal{R}} g(a, b) \rightarrow_{\mathcal{R}} f(a) \rightarrow_{\mathcal{R}} \dots$ while collapsed tree rewriting transforms $\blacktriangle f(a)$ in two steps into $\blacktriangle g(b, b)$.

⁷Another counterexample can be found in Chapter 8, see Example 8.14.

Chapter 7

Critical Pairs

In Section 6.5, confluence properties are investigated by relating collapsed tree rewriting to term rewriting. The motivation for the present chapter is to give a sufficient condition for local confluence of collapsed tree rewriting directly in terms of properties of evaluation and folding rules. The idea is, in analogy to the well-known results for term rewriting established by Knuth and Bendix [KB70] and Huet [Hue80], to infer local confluence of $\Rightarrow_{\mathcal{R}}$ from the confluence of so-called *critical pairs*, being certain divergent steps $T \Leftarrow S \Rightarrow U$ in which S represents a “critical overlap” of the left-hand sides of the two applied rules.

It turns out that divergent steps of the form $D \Leftarrow_{\mathcal{E}} C \Rightarrow_{\mathcal{F}} \tilde{C}$ are always confluent, independently of the given rewrite system \mathcal{R} . Thus, since folding is also confluent, only those divergent steps are critical in which S is obtained by the superposition of the left-hand sides of two evaluation rules. The main result of this chapter, the *Critical Pair Lemma*, shows that $\Rightarrow_{\mathcal{R}}$ is locally confluent if all critical pairs are confluent in a specific way. This result leads—in the presence of termination—to a characterization of confluence through critical pairs and to a decision procedure for confluence.

7.1 Characterizing Local Confluence

This section prepares the ground for the proof of the Critical Pair Lemma by showing that $\Rightarrow_{\mathcal{R}}$ is locally confluent if and only if all divergent $\Rightarrow_{\mathcal{E}}$ -steps are confluent under $\Rightarrow_{\mathcal{R}}$. This requires to show that the collapsed trees produced by divergent evaluation and folding steps always have a common reduct.

The proof of the latter is based on the Soundness Theorem and uses two lemmas that determine how the sets of term positions corresponding to certain nodes are changed by evaluation and folding steps. To establish these lemmas, it is convenient to characterize positions by paths.

Given a node v in a collapsed tree C , denote by $PATH_C(v)$ the set of all paths

from $root_C$ to v . Moreover, for a (possibly empty) path $p = \langle e_1, i_1 \rangle, \dots, \langle e_n, i_n \rangle$, let $\alpha(p)$ be the string $i_1 \dots i_n$.

Lemma 7.1 *For every collapsed tree C and each node v in C ,*

$$node_C^{-1}(v) = \{\alpha(q) \mid q \in PATH_C(v)\}.$$

Proof For each q in $PATH_C(v)$, induction on the length of q shows that $\alpha(q)$ is a position in $term(C)$ such that $node_C(\alpha(q)) = v$. Conversely, given a position $i_1 \dots i_n$ in $node_C^{-1}(v)$, induction on n yields a path $\langle e_1, i_1 \rangle, \dots, \langle e_n, i_n \rangle$ from $root_C$ to v . \square

In applying an evaluation rule ($\underline{L} \leftarrow \underline{K} \rightarrow \underline{R}$) to a collapsed tree C , nodes that are not reachable from the image of $root_L$ keep their associated term positions.

Lemma 7.2 *Let $C \Rightarrow_{p,g} D$ be an evaluation step through $p = (\underline{L} \leftarrow \underline{K} \rightarrow \underline{R})$, and let v be a node in C such that $g_V(root_L) \not\prec_C v$. Then*

$$node_D^{-1}(track(v)) = node_C^{-1}(v).$$

Proof Let $C \Rightarrow_{p,g} M$ underly $C \Rightarrow_{p,g} D$, with the following associated diagram:

$$\begin{array}{ccccc} \underline{L} & \longleftarrow & \underline{K} & \xrightarrow{b} & \underline{R} \\ g \downarrow & & d \downarrow & & \downarrow h \\ C & \longleftarrow & J & \xrightarrow{c} & M \end{array}$$

The proposition clearly holds for $v = root_C$; assume therefore $v \neq root_C$.

By Lemma 7.1, each position in $node_C^{-1}(v)$ can be written as $\alpha(q)$ for some q in $PATH_C(v)$. Since J is obtained from C by removing the edge with source $g_V(root_L)$, $g_V(root_L) \not\prec_C v$ implies that this edge does not belong to q . So q is a path in J and hence c maps q to a path q' in M from $c_V(root_C)$ to $c_V(v)$ and such that $\alpha(q') = \alpha(q)$; q' is in $PATH_D(track(v))$ since $c_V(root_C) = track(root_C) = root_D$ and $c_V(v) = track(v)$. With Lemma 7.1 follows $\alpha(q) = \alpha(q') \in node_D^{-1}(track(v))$.

Conversely, consider some q in $PATH_D(track(v))$, that is, q is a path from $c_V(root_C)$ to $c_V(v)$.

Assumption: q contains an edge that is not in cJ . Then there is a component $\langle e, i \rangle$ in q such that $e \notin E_{cJ}$ but $t_M(e)|_i \in V_{cJ}$. Let $w = t_M(e)|_i$. One obtains $w \geq_{cJ} c_V(v)$ since each path from w to $c_V(v)$ lies in cJ (otherwise such a path contained the edge outgoing from $c_V(g_V(root_L))$), and hence $w >_M c_V(g_V(root_L)) >_M w$ in contradiction to the acyclicity of M . Also, by Lemmas 5.6 and 3.6(4), c is injective because $M - cJ$ (and hence $\underline{R} - b\underline{K}$) is nonempty. It follows $x \geq_J v$ and

$x \geq_C v$, where x is the unique node with $c_V(x) = w$. Moreover, as w is a target node of an edge in $M - cJ = \underline{R} - b\underline{K}$, there is an open node in \underline{R} that is mapped by h to w . Let \bar{x} be the preimage of this open node in \underline{K} . Then $c_V(d_V(\bar{x})) = h_V(b_V(\bar{x})) = w$ and hence $d_V(\bar{x}) = x$. So in particular $g_V(\bar{x}) = x$. But as there is a path from $root_L$ to each other node in \underline{L} , one obtains $g_V(root_L) >_C x$. Therefore $g_V(root_L) >_C x \geq_C v$, contradicting the assumption of the lemma. Thus the assumption that q contains an edge from $M - cJ$ is wrong, that is, q is a path in cJ . Furthermore, $c_V(g_V(root_L))$ is not a source node of any edge in q , as otherwise there were a node y in C such that $c_V(y) = c_V(g_V(root_L))$ and $g_V(root_L) >_C y \geq_C v$. Since $g_V(root_L)$ is the only node that may be identified with some other node by c , it follows that the preimages of the edges in q form a path q' in J from $root_C$ to v such that $\alpha(q') = \alpha(q)$. So $q' \in PATH_C(v)$ and, by Lemma 7.1, $\alpha(q) = \alpha(q') \in node_{\tilde{C}}^{-1}(v)$. \square

The next lemma is the analogue of Lemma 7.2 for folding steps. In this case the set of positions associated with a node v changes if and only if v is one of the two nodes that are identified.

Lemma 7.3 *Let $C \Rightarrow_{\mathcal{F}} \tilde{C}$ be a folding step between collapsed trees, and let v_1 and v_2 be the two nodes in C that are identified. Then for each node v in C ,*

$$node_{\tilde{C}}^{-1}(track(v)) = \begin{cases} node_{\tilde{C}}^{-1}(v_1) \cup node_{\tilde{C}}^{-1}(v_2) & \text{if } v \in \{v_1, v_2\}, \\ node_{\tilde{C}}^{-1}(v) & \text{otherwise.} \end{cases}$$

Proof By Lemma 4.16 there is a surjective jungle morphism $f: C \rightarrow \tilde{C}$ such that $f_V = track$. Let v be a node in C . If $v = root_C$, then $v \notin \{v_1, v_2\}$ and $node_{\tilde{C}}^{-1}(f_V(v)) = node_{\tilde{C}}^{-1}(root_{\tilde{C}}) = \{\lambda\} = node_{\tilde{C}}^{-1}(v)$. Assume therefore $v \neq root_C$. Define a function

$$\psi: \bigcup \{PATH_C(v') \mid f_V(v') = f_V(v)\} \rightarrow PATH_{\tilde{C}}(f_V(v))$$

by $\langle e_1, i_1 \rangle, \dots, \langle e_n, i_n \rangle \mapsto \langle f_E(e_1), i_1 \rangle, \dots, \langle f_E(e_n), i_n \rangle$. This function is surjective because f_E is.

Case 1: $v \in \{v_1, v_2\}$. Then ψ has $PATH_C(v_1) \cup PATH_C(v_2)$ as domain. Hence, by Lemma 7.1,

$$\begin{aligned} node_{\tilde{C}}^{-1}(v_1) \cup node_{\tilde{C}}^{-1}(v_2) &= \alpha(PATH_C(v_1)) \cup \alpha(PATH_C(v_2)) \\ &= \alpha(\psi(PATH_C(v_1))) \cup \alpha(\psi(PATH_C(v_2))) \\ &= \alpha(\psi(PATH_C(v_1) \cup PATH_C(v_2))) \\ &= \alpha(PATH_{\tilde{C}}(f_V(v))) && \psi \text{ surj.} \\ &= node_{\tilde{C}}^{-1}(f_V(v)). \end{aligned}$$

Case 2: $v \notin \{v_1, v_2\}$. Then ψ has $PATH_C(v)$ as domain and hence, similar to Case 1, Lemma 7.1 and surjectivity of ψ allow to conclude $node_{\tilde{C}}^{-1}(v) = node_{\tilde{C}}^{-1}(f_V(v))$. \square

Now the desired result can be established that collapsed trees resulting from two divergent evaluation and folding steps always have a common reduct. The crucial case for the proof is given when the edge removed by the evaluation step is one of the two edges to which the folding rule is applied. This case is illustrated by the following example.

Example 7.4 Consider the rewrite rule $f(a) \rightarrow g(b)$ and the collapsed tree in the middle of the upper row in Figure 7.1. The evaluation step on the left replaces one of two f -labelled edges which are subject to the folding step on the right. The collapsed trees produced by these steps are then rewritten by applying the evaluation rule for $f(a) \rightarrow g(b)$ again. This results in two collapsed trees representing the same term (viz. $h(g(b), g(b))$), so that subsequent folding steps lead to a common reduct.

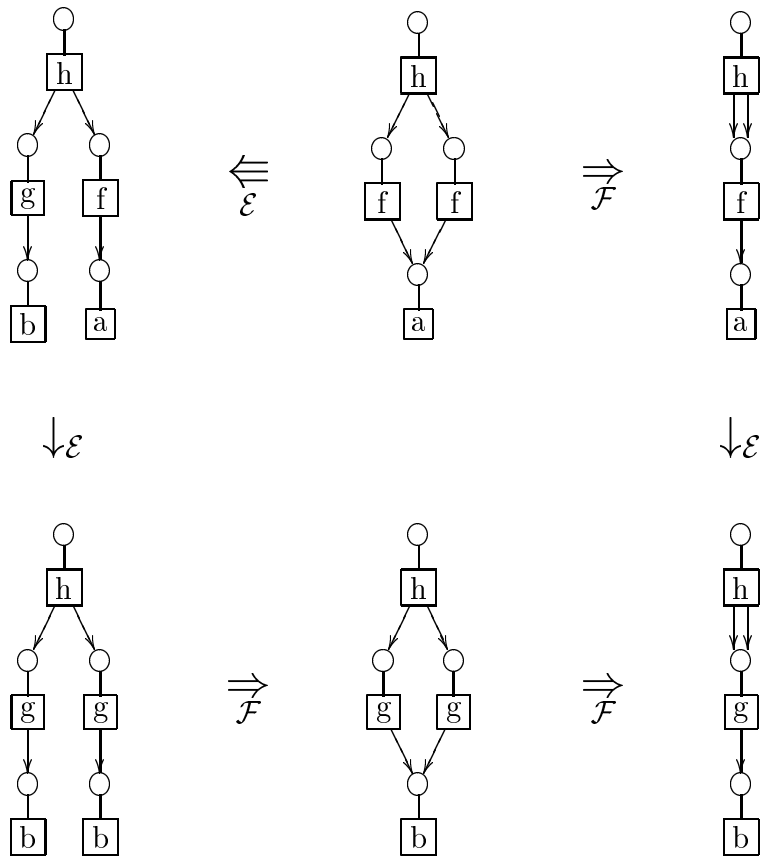


Figure 7.1: Confluence in the case of divergent evaluation and folding steps

Theorem 7.5 *The relation $\Rightarrow_{\mathcal{R}}$ is locally confluent if and only if for all evaluation steps $D \Leftarrow_{\varepsilon} C \Rightarrow_{\varepsilon} F$ there is a collapsed tree X such that $D \Rightarrow_{\mathcal{R}}^* X \Leftarrow_{\mathcal{R}}^* F$.*

Proof As the “only if” direction is obvious, assume that the second property in the proposition is satisfied. Then, since folding is confluent (Theorem 4.18), it suffices to consider two steps of the form $D \Leftarrow_{\varepsilon} C \Rightarrow_{\mathcal{F}} \tilde{C}$ and to show that D and \tilde{C} have a common reduct under $\Rightarrow_{\mathcal{R}}$. Let $C \Rightarrow_{p,g} M$ be the evaluation step underlying $C \Rightarrow_{\varepsilon} D$, based on a rewrite rule $l \rightarrow r$. By Lemma 4.16 there is a jungle morphism $f: C \rightarrow \tilde{C}$. So there is an evaluation step $\tilde{C} \Rightarrow_{p,f \circ g} B$, since every jungle morphism outgoing from the left-hand side of an evaluation rule satisfies the gluing condition (see the proof of Theorem 5.21). By the Soundness Theorem 5.14,

$$\text{term}(C) = \text{term}(\tilde{C}) \Rightarrow_{\Delta} \text{term}(B) \quad (7.1)$$

through $l \rightarrow r$, where $\Delta = \text{node}_{\tilde{C}}^{-1}(f_V(g_V(\text{root}_{\diamond l})))$. Moreover,

$$\text{term}(C) \Rightarrow_{\Gamma} \text{term}(D) \quad (7.2)$$

through $l \rightarrow r$, with $\Gamma = \text{node}_C^{-1}(g_V(\text{root}_{\diamond l}))$. Let v_1, v_2 be the two nodes in C that are identified by f .

Case 1: $g_V(\text{root}_{\diamond l}) \notin \{v_1, v_2\}$. By Lemma 4.16, $f_V = \text{track}_{C \rightarrow \tilde{C}}$. With Lemma 7.3 follows $\Delta = \text{node}_{\tilde{C}}^{-1}(g_V(\text{root}_{\diamond l}))$. So $\Delta = \Gamma$ and hence $\text{term}(B) = \text{term}(D)$. Thus $B \Rightarrow_{\mathcal{F}}^* \blacktriangle \text{term}(B) \Leftarrow_{\mathcal{F}}^* D$ by Lemma 6.11.

Case 2: $g_V(\text{root}_{\diamond l}) \in \{v_1, v_2\}$. Let $C \Rightarrow_{p,g} M$ have the following diagram:

$$\begin{array}{ccccc} \underline{L} & \longleftarrow & \underline{K} & \longrightarrow & \underline{R} \\ g \downarrow & & \downarrow & & \downarrow \\ C & \longleftarrow & J & \xrightarrow{c} & M \end{array}$$

Assume without loss of generality $g_V(\text{root}_{\diamond l}) = v_1$. Since v_1 and v_2 are the source nodes of two edges that have identical labels and target strings, there is an isomorphism $i: C/v_1 \rightarrow C/v_2$ that maps v_1 to v_2 . Also, $C/v_2 \subseteq J$ because J is obtained from C by removing the edge with source v_1 . Hence, composing the restriction of g to C/v_1 with i and the restriction of c to C/v_2 yields a jungle morphism $h: \underline{L} \rightarrow M$ that maps $\text{root}_{\diamond l}$ to $c_V(v_2)$. Moreover, $\text{root}_C >_J v_2$ implies $c_V(\text{root}_C) >_M c_V(v_2)$, and therefore $M/c_V(v_2) \subseteq M/c_V(\text{root}_C) = D$. So $h\underline{L} \subseteq M/c_V(v_2) \subseteq D$ and consequently h has a restriction $h': \underline{L} \rightarrow D$. Thus there is an evaluation step $D \Rightarrow_{p,h'} F$. By the Soundness Theorem,

$$\text{term}(D) \Rightarrow_{\Lambda} \text{term}(F) \quad (7.3)$$

through $l \rightarrow r$, where $\Lambda = \text{node}_D^{-1}(h'_V(\text{root}_{\diamond l}))$. Since $\Lambda = \text{node}_D^{-1}(c_V(v_2)) = \text{node}_D^{-1}(\text{track}_{C \Rightarrow_{\varepsilon} D}(v_2))$, Lemma 7.2 yields $\Lambda = \text{node}_C^{-1}(v_2)$ (note that $v_1 \not\prec_C v_2$).

Also, by Lemma 7.3, $\Delta = \text{node}_{\tilde{C}}^{-1}(\text{track}_{C \Rightarrow \tilde{C}}(v_1)) = \text{node}_{\tilde{C}}^{-1}(v_1) \cup \text{node}_{\tilde{C}}^{-1}(v_2)$. Hence $\Delta = \Gamma \cup \Lambda$. Thus, since

$$\text{term}(B) \Leftarrow_{\Delta} \text{term}(C) \Rightarrow_{\Gamma} \text{term}(D) \Rightarrow_{\Lambda} \text{term}(F)$$

by (7.1) to (7.3) and $\Gamma \cap \Lambda = \emptyset$, it follows $\text{term}(B) = \text{term}(F)$. Then, finally, $B \Rightarrow_{\mathcal{F}}^* \blacktriangle \text{term}(B) \Leftarrow_{\mathcal{F}}^* F$ by Lemma 6.11. \square

The following corollary is an immediate consequence of Theorem 7.5.

Corollary 7.6 *If $\Rightarrow_{\varepsilon}$ is locally confluent, then so is $\Rightarrow_{\mathcal{R}}$.*

While Theorem 7.5 provides a characterization of local confluence of collapsed tree rewriting, Corollary 7.6 gives only a sufficient condition which is not necessary. As an example, consider the convergent term rewriting system

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

for which $\Rightarrow_{\mathcal{R}}$ is convergent by Corollary 6.23. That $\Rightarrow_{\varepsilon}$ is not locally confluent can be seen by the steps $\blacktriangle g(a, a) \Leftarrow_{\varepsilon} \blacktriangle f(a) \Rightarrow_{\varepsilon} \Delta g(a, b)$: the only collapsed trees derivable from $\blacktriangle g(a, a)$ and $\Delta g(a, b)$ by $\Rightarrow_{\varepsilon}$ -steps are $\blacktriangle g(b, b)$ respectively $\Delta g(b, b)$; since the latter are non-isomorphic, $\Rightarrow_{\varepsilon}$ is not locally confluent.

Note also that it is not possible to guarantee confluence of $\Rightarrow_{\mathcal{R}}$ by strengthening the condition of Corollary 7.6 to confluence or even strong confluence of $\Rightarrow_{\varepsilon}$. The simplest counterexample is provided by the one-rule system $a \rightarrow f(a)$: here $\Rightarrow_{\varepsilon}$ can be shown to be strongly confluent by Theorem 3.8, but Example 5.28 demonstrates that $\Rightarrow_{\mathcal{R}}$ is non-confluent in the presence of a binary function symbol.

7.2 The Critical Pair Lemma

According to Theorem 7.5, testing collapsed tree rewriting for local confluence reduces to the problem to check that every pair of divergent evaluation steps is confluent under $\Rightarrow_{\mathcal{R}}$. The Critical Pair Lemma established below provides a sufficient condition for the latter showing that it suffices to consider only those divergent steps $U_1 \Leftarrow_{\varepsilon} S \Rightarrow_{\varepsilon} U_2$ in which S represents a “critical overlap” of the left-hand sides of the applied evaluation rules. The requirement is then that there are derivations $U_1 \Rightarrow_{\varepsilon \cup \mathcal{F}}^* X_1$ and $U_2 \Rightarrow_{\varepsilon \cup \mathcal{F}}^* X_2$ such that X_1 and X_2 are isomorphic up to certain garbage by an isomorphism that is compatible with the track functions of the derivations $S \Rightarrow_{\varepsilon} U_1 \Rightarrow_{\varepsilon \cup \mathcal{F}}^* X_1$ and $S \Rightarrow_{\varepsilon} U_2 \Rightarrow_{\varepsilon \cup \mathcal{F}}^* X_2$.

The proof of the Critical Pair Lemma is based on the Extension Lemma 6.2 which allows to extend the confluent derivations of critical pairs by context. In the Extension Lemma, the resulting jungle of the extended derivation is constructed by a pushout. This is exploited, roughly speaking, to show that the results of the two extended derivations of a critical pair are isomorphic up to certain garbage.

As the Extension Lemma is about derivations without garbage collection, critical pairs are also defined for $\Rightarrow_{\mathcal{E}}$ -steps and not for $\Rightarrow_{\mathcal{E}}$ -steps. (An adaptation of the Extension Lemma to collapsed tree rewriting is not possible since context edges may be attached to nodes that become garbage, preventing the construction of extended collapsed trees by pushouts.)

In the following definition, the first condition puts a bound on the size of collapsed trees in critical pairs which depends only on the given set \mathcal{E} of evaluation rules. The second condition is derived from Theorem 3.8 and expresses that left-hand sides of evaluation rules overlap in a critical way.

Definition 7.7 (critical pair) Let $p_i = (L_i \leftarrow K_i \rightarrow R_i)$ be an evaluation rule, for $i = 1, 2$. A pair of direct derivations of the form $U_1 \leftarrow_{p_1, g_1} S \Rightarrow_{p_2, g_2} U_2$ is a *critical pair* (over \mathcal{E}) if

- (1) S is a jungle such that $S = g_1 L_1 \cup g_2 L_2$ and
- (2) $g_1 L_1 \cap g_2 L_2 \neq g_1 K_1 \cap g_2 K_2$.

Moreover, $g_1 \neq g_2$ is required for the case $p_1 = p_2$.

By the construction of evaluation rules, condition (2) is equivalent to “ $g_{1E}(e) \in g_{2E}E_{L_2}$ or $g_{2E}(e_2) \in g_{1E}E_L$ ”, where e_1 and e_2 are the unique edges in $L_1 - K_1$ respectively $L_2 - K_2$. Note also that the jungles S , U_1 , and U_2 are not closed in general.¹ In what follows, critical pairs that differ only by renaming of nodes and edges are not distinguished. As a consequence, only finitely many critical pairs need to be considered whenever \mathcal{R} has a finite rule set.

Example 7.8 The rewrite rules

$$\begin{array}{l} g(f(a, x)) \rightarrow g(b) \\ f(x, y) \rightarrow c \end{array}$$

give rise to the two critical pairs shown in Figure 7.2 (the nodes are numbered to indicate the track functions).

¹Straightforward adaptations of the proofs of Lemmas 5.7 and 4.16 show that evaluation and folding rules preserve also non-closed jungles.

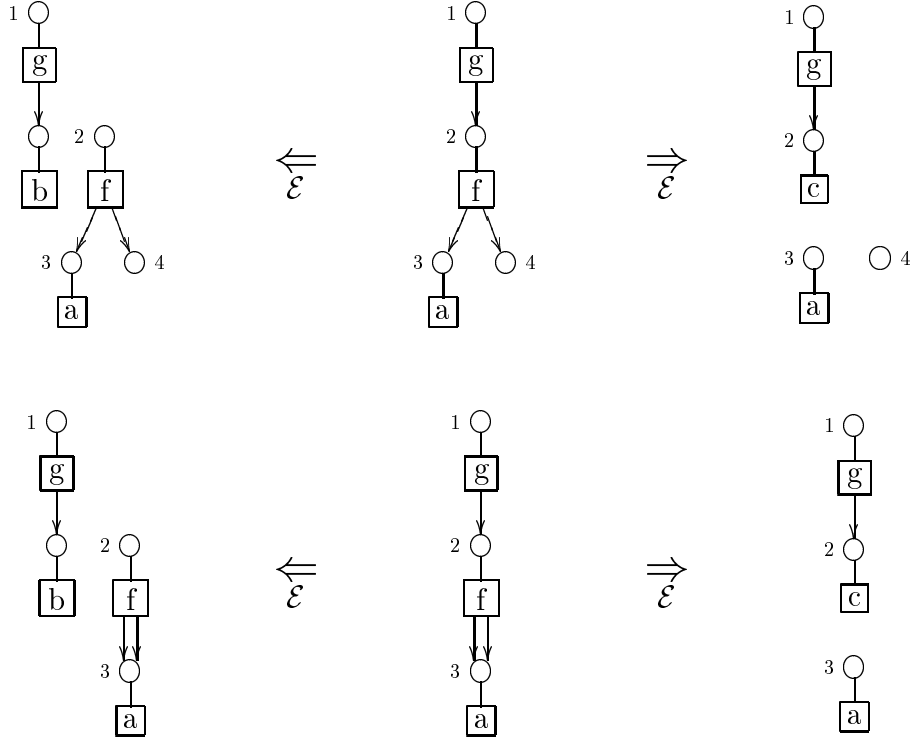


Figure 7.2: Two critical pairs

Definition 7.9 (joinability) A critical pair $U_1 \leftarrow_{\mathcal{E}} S \Rightarrow_{\mathcal{E}} U_2$ is *joinable* if there are derivations $U_1 \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_1$ and $U_2 \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_2$ such that there is an isomorphism $iso: X_1^\bullet \rightarrow X_2^\bullet$ with $iso_V \circ tr_1 = tr_2$, where X_i^\bullet is the subjungle of X_i induced by $\{x \mid track_{S \Rightarrow U_i \Rightarrow^* X_i}(v) \geq_{X_i} x \text{ for some } v \in V_S\}$ and tr_i is the restriction of $track_{S \Rightarrow U_i \Rightarrow^* X_i}$ to $V_{X_i^\bullet}$, for $i = 1, 2$.

Note that the joining derivations $U_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_i$ are allowed to contain folding steps and that garbage in X_1 and X_2 is ignored as far as it consists of items not reachable from the descendant of any node in S . Requiring that X_1 and X_2 are isomorphic would be too restrictive, as can be seen in the following example.

Example 7.10 After adding the rewrite rule $g(b) \rightarrow g(c)$ to the two rules of Example 7.8, the second critical pair in Figure 7.2 is joinable by the derivation shown in Figure 7.3. Observe that the resulting jungle contains garbage (the constant b) which does not occur in the right-hand jungle of the critical pair.

The first critical pair in Figure 7.2 is joinable by a similar derivation.

Now the main result of this chapter can be stated.

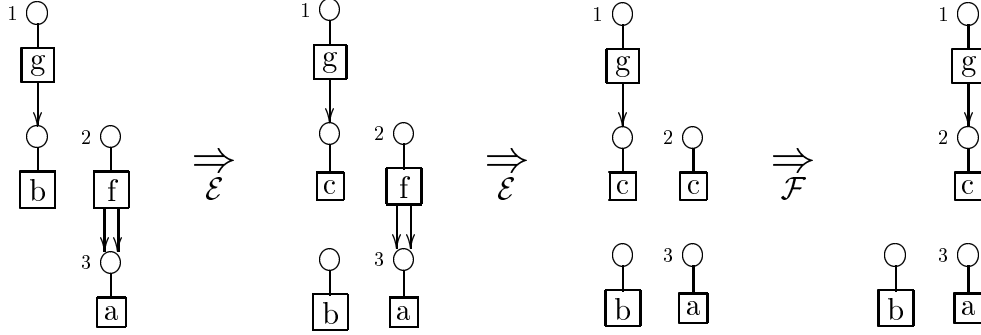


Figure 7.3: A joining derivation for Example 7.8

Lemma 7.11 (Critical Pair Lemma) *If all critical pairs over \mathcal{E} are joinable, then $\Rightarrow_{\mathcal{R}}$ is locally confluent.*

Proof By Theorem 7.5 it suffices to show that for every pair $D_1 \Leftarrow_{\mathcal{E}} C \Rightarrow_{\mathcal{E}} D_2$ of evaluation steps, D_1 and D_2 have a common reduct under $\Rightarrow_{\mathcal{R}}$. Consider evaluation steps $M_1 \Leftarrow_{p_1, g_1} C \Rightarrow_{p_2, g_2} M_2$ such that $D_i = M_i / \text{track}_{C \Rightarrow M_i}(\text{root}_C)$ for $i = 1, 2$, and let $p_i = (L_i \leftarrow K_i \rightarrow R_i)$ for $i = 1, 2$.

Case 1: $g_1 L_1 \cap g_2 L_2 = g_1 K_1 \cap g_2 K_2$. Then, by Theorem 3.8, there is a jungle M such that $M_1 \Rightarrow_{p_2} M \Leftarrow_{p_1} M_2$ and $\text{track}_{C \Rightarrow M_1 \Rightarrow M} = \text{track}_{C \Rightarrow M_2 \Rightarrow M}$. By the Translation Lemma 6.8, $D_i \Rightarrow_{\mathcal{R}}^{\lambda} M / \text{track}_{M_i \Rightarrow M}(\text{root}_{D_i})$ for $i = 1, 2$. So

$$D_1 \Rightarrow_{\mathcal{R}}^{\lambda} M / \text{track}_{M_1 \Rightarrow M}(\text{root}_{D_1}) \Leftarrow_{\mathcal{R}}^{\lambda} D_2$$

since $\text{track}_{M_1 \Rightarrow M}(\text{root}_{D_1}) = \text{track}_{C \Rightarrow M_1 \Rightarrow M}(\text{root}_C) = \text{track}_{C \Rightarrow M_2 \Rightarrow M}(\text{root}_C) = \text{track}_{M_2 \Rightarrow M}(\text{root}_{D_2})$.

Case 2: $g_1 L_1 \cap g_2 L_2 \neq g_1 K_1 \cap g_2 K_2$. Assume that $p_1 \neq p_2$ or $g_1 \neq g_2$, as otherwise D_1 and D_2 are isomorphic and trivially possess a common reduct. Let $S = g_1 L_1 \cup g_2 L_2$. By the Restriction Lemma 6.1 there are direct derivations $U_1 \Leftarrow_{p_1, g'_1} S \Rightarrow_{p_2, g'_2} U_2$ such that for $i = 1, 2$, g'_i is restriction of g_i , $U_i \subseteq M_i$, and $\text{track}_{S \Rightarrow U_i}$ is restriction of $\text{track}_{C \Rightarrow M_i}$. Clearly these two derivations constitute a critical pair. As critical pairs are joinable by assumption, there are derivations $U_1 \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_1$ and $U_2 \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_2$ such that there is an isomorphism $iso: X_1^{\bullet} \rightarrow X_2^{\bullet}$ with $iso \circ tr_1 = tr_2$, where $X_i^{\bullet} \subseteq X_i$ is induced by

$$\{x \mid \text{track}_{S \Rightarrow U_i \Rightarrow^* X_i}(v) \geq_{X_i} x \text{ for some } v \in V_S\}$$

and tr_i is the restriction of $\text{track}_{S \Rightarrow U_i \Rightarrow^* X_i}$ to $V_{X_i^{\bullet}}$, for $i = 1, 2$. Since S is included in C , the Extension Lemma 6.2 shows that $S \Rightarrow_{\mathcal{E}} U_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_i$ extends to a

derivation $C \Rightarrow_{\mathcal{E}} M'_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* N_i$ such that the diagram

$$\begin{array}{ccc} V & \xrightarrow{\widehat{tr}_i} & X_i \\ \text{incl} \downarrow & [1] & \downarrow h_i \\ J & \xrightarrow{c_i} & N_i \end{array}$$

is a pushout, where V is V_S considered as a jungle, $J \subseteq C$ has node set V_C and edge set $E_C - E_S$, incl is the inclusion of V in J , \widehat{tr}_i is $\text{track}_{S \Rightarrow U_i \Rightarrow^* X_i}$ considered as a jungle morphism, and $c_{iV} = \text{track}_{C \Rightarrow^* N_i}$, for $i = 1, 2$. Moreover, by the proof of the Extension Lemma 6.1, C derives M'_i through p_i and g_i , so $M'_i \cong M_i$ and

$$M'_i / \text{track}_{C \Rightarrow M'_i}(\text{root}_C) \cong D_i \quad (i = 1, 2). \quad (7.4)$$

Now decompose square [1] as shown below,

$$\begin{array}{ccccc} V & \xrightarrow{tr_i} & X_i^\bullet & \longrightarrow & X_i \\ \text{incl} \downarrow & [2] & h'_i \downarrow & [3] & \downarrow h_i \\ J & \xrightarrow{c'_i} & N'_i & \longrightarrow & N_i \end{array}$$

where tr_i is considered as a jungle morphism, $X_i^\bullet \rightarrow X_i$ is inclusion, N'_i is the subjungle of N_i with node and edge sets $N_i - (h_i X_i - h_i X_i^\bullet)$, $N'_i \rightarrow N_i$ is inclusion, and h'_i, c'_i are restrictions of h_i respectively c_i , for $i = 1, 2$. By Lemma 3.5, N'_i is well-defined and square [3] is a pushout, provided that h_i satisfies the gluing condition. Injectivity of h_i holds by Lemma 3.6(4), since [1] is a pushout. To see that the contact condition is satisfied, observe that $h_i X_i - h_i X_i^\bullet = h_i(X_i - X_i^\bullet) \subseteq h_i(X_i - \widehat{tr}_i V)$ because the nodes in $\widehat{tr}_i V$ belong to X_i^\bullet ; but $h_i(X_i - \widehat{tr}_i V) \cap c_i J = \emptyset$ by Lemma 3.6(3) and hence no edge in $N_i - h_i X_i \subseteq c_i J$ is incident to any node in $h_i X_i - h_i X_i^\bullet$.

Also, $(h_i X_i - h_i X_i^\bullet) \cap c_i J = \emptyset$ implies that c'_i is well-defined. Since [3] and the outer rectangle $[2] \cup [3] = [1]$ are pushouts, square [2] is a pushout by Lemma 3.7(3). Now consider the diagram

$$\begin{array}{ccccc} V & \xrightarrow{tr_1} & X_1^\bullet & \xrightarrow{iso} & X_2^\bullet \\ \text{incl} \downarrow & [2'] & h'_1 \downarrow & [4] & \downarrow \\ J & \xrightarrow{c'_1} & N'_1 & \xrightarrow{iso_1} & N \end{array}$$

in which $[2']$ is $[2]$ instantiated with $i = 1$, and $[4]$ is the pushout of h'_1 and iso . By assumption, $iso \circ tr_1 = tr_2$. The commutativity of $[2'']$, being $[2]$ instantiated with $i = 2$, implies $c'_2 \circ incl = h'_2 \circ tr_2 = h'_2 \circ iso \circ tr_1$. Hence, because the outer rectangle $[2'] \cup [4]$ is a pushout, there is a jungle morphism $iso_2: N \rightarrow N'_2$ such that

$$iso_2 \circ iso_1 \circ c'_1 = c'_2. \quad (7.5)$$

Moreover, as both $[2'] \cup [4]$ and $[2'']$ are pushouts of $incl$ and tr_2 , iso_2 is an isomorphism by Lemma 3.6(2). Also, by Lemma 3.6(4), iso_1 is an isomorphism because it lies opposite to iso in pushout $[4]$. So $iso' = iso_2 \circ iso_1$ is an isomorphism from N'_1 to N'_2 . Thus, letting $v_i = track_{C \Rightarrow^* M'_i \Rightarrow^* N_i}(root_C)$ for $i = 1, 2$, one obtains the following:

$$\begin{aligned} N_1/v_1 &= N'_1/v_1 && \text{construction of } N'_1 \\ &\cong iso'(N'_1/v_1) && \\ &= N'_2/iso'_V(v_1) && \text{Lemma 6.4} \\ &= N'_2/iso'_V(c_{1V}(root_C)) && \text{Extension Lemma} \\ &= N'_2/iso'_V(c'_{1V}(root_C)) && \text{construction of } c'_1 \\ &= N'_2/c'_{2V}(root_C) && (7.5) \\ &= N'_2/c_{2V}(root_C) && \text{construction of } c'_2 \\ &= N'_2/v_2 && \text{Extension Lemma} \\ &= N_2/v_2 && \text{construction of } N'_2 \end{aligned}$$

Also, the Translation Lemma 6.8 yields $M'_i/track_{C \Rightarrow M'_i}(root_C) \Rightarrow_{\mathcal{R}}^* N_i/v_i$ since $C \Rightarrow_{\mathcal{E} \cup \mathcal{F}} M'_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* N_i$ ($i = 1, 2$). So

$$M'_1/track_{C \Rightarrow M'_1}(root_C) \Rightarrow_{\mathcal{R}}^* N_1/v_1 \cong N_2/v_2 \Leftarrow_{\mathcal{R}}^* M'_2/track_{C \Rightarrow M'_2}(root_C)$$

and hence, by (7.4), $D_1 \Rightarrow_{\mathcal{R}}^* N_1/v_1 \Leftarrow_{\mathcal{R}}^* D_2$. Thus $\Rightarrow_{\mathcal{R}}$ is locally confluent. \square

It is open whether the converse of the Critical Pair Lemma also holds, that is, whether local confluence of $\Rightarrow_{\mathcal{R}}$ implies that all critical pairs are joinable. The problem is to show that there are always joining derivations such that the associated track functions are compatible in the way required in Definition 7.9.

Nevertheless, the Critical Pair Lemma allows to characterize confluence in the presence of termination, analogously to the well-known result of Knuth and Bendix [KB70] for term rewriting. For proving the characterization, derivations over variable-collapsed jungles are restricted to jungles without variables.

Lemma 7.12 (removal of variables) *Let $G \Rightarrow_{p,g} H$ be an evaluation or folding step between variable-collapsed jungles. Then there is a direct derivation $\underline{G} \Rightarrow_{p,g'} \underline{H}$ such that g' is restriction of g and $track_{\underline{G} \Rightarrow \underline{H}} = track_{G \Rightarrow H}$.*

Proof The image of the left-hand side of p in G contains no variable-edges since such edges do not appear in evaluation rules and are not foldable in variable-collapsed jungles. Hence, by the Restriction Lemma 6.1, there is a direct derivation $\underline{G} \Rightarrow_{p, g'} W$ with $W \subseteq H$ and such that $track_{\underline{G} \Rightarrow W}$ is restriction of $track_{G \Rightarrow H}$. The proof of the Restriction Lemma yields $W = \underline{H}$. Then $track_{\underline{G} \Rightarrow \underline{H}} = track_{G \Rightarrow H}$ since $V_{\underline{G}} = V_G$ and $V_{\underline{H}} = V_H$. \square

Theorem 7.13 *Suppose that $\Rightarrow_{\mathcal{R}}$ is terminating. Then $\Rightarrow_{\mathcal{R}}$ is confluent if and only if all critical pairs over \mathcal{E} are joinable.*

Proof If all critical pairs are joinable, then $\Rightarrow_{\mathcal{R}}$ is locally confluent by the Critical Pair Lemma and hence confluent by Newman's Lemma (see page 10).

To show the converse, assume that $\Rightarrow_{\mathcal{R}}$ is confluent and consider any critical pair $U_1 \leftarrow_{p_1, g_1} S \Rightarrow_{p_2, g_2} U_2$. In the rest of the proof it is shown that this critical pair is joinable.

Extend S to a variable-collapsed jungle \hat{S} with $\hat{\underline{S}} = S$, by appending edges labelled with variables to the open nodes of S such that no variable occurs twice. Then, by the proof of the Extension Lemma 6.2, there are evaluation steps $\hat{U}_1 \leftarrow_{p_1, \hat{g}_1} \hat{S} \Rightarrow_{p_2, \hat{g}_2} \hat{U}_2$ such that, for $i = 1, 2$, \hat{g}_i is the extension of g_i to \hat{S} , and $\hat{U}_i = U_i$.

By Theorem 6.17, $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is terminating since $\Rightarrow_{\mathcal{R}}$ is. So there is a derivation $\hat{U}_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* \hat{X}_i$ such that \hat{X}_i is a normal form, for $i = 1, 2$. Let \hat{X}_i^\bullet be the subjungle of \hat{X}_i induced by

$$\{v \mid track_{\hat{S} \Rightarrow \hat{U}_i \Rightarrow^* \hat{X}_i}(\bar{v}) \geq_{\hat{X}_i} v \text{ for some } \bar{v} \in V_S\},$$

for $i = 1, 2$. Since $\rightarrow_{\mathcal{R}}$ is confluent by Theorem 6.19, one obtains $TERM_{\hat{X}_1^\bullet} = TERM_{\hat{X}_2^\bullet}$ similarly as in the proof of Theorem 5.32. Hence, by Theorem 4.13, there is an isomorphism $\widehat{iso}: \hat{X}_1^\bullet \rightarrow \hat{X}_2^\bullet$. Now consider any node \bar{v} in \hat{S} , and let $v_i = track_{\hat{S} \Rightarrow \hat{U}_i \Rightarrow^* \hat{X}_i}(\bar{v})$ for $i = 1, 2$. By Corollaries 5.18 and 5.23, $term_{\hat{X}_1}(v_1)$ and $term_{\hat{X}_2}(v_2)$ are normal forms of $term_{\hat{S}}(\bar{v})$. Then $term_{\hat{X}_1}(v_1) = term_{\hat{X}_2}(v_2)$ by confluence of $\rightarrow_{\mathcal{R}}$, and hence

$$\begin{aligned} term_{\hat{X}_2^\bullet}(\widehat{iso}_V(track_{\hat{S} \Rightarrow \hat{U}_1 \Rightarrow^* \hat{X}_1}(\bar{v}))) &= term_{\hat{X}_2^\bullet}(\widehat{iso}_V(v_1)) \\ &= term_{\hat{X}_1^\bullet}(v_1) && \text{Lemma 4.6} \\ &= term_{\hat{X}_1}(v_1) \\ &= term_{\hat{X}_2}(v_2) \\ &= term_{\hat{X}_2^\bullet}(v_2) \\ &= term_{\hat{X}_2^\bullet}(track_{\hat{S} \Rightarrow \hat{U}_2 \Rightarrow^* \hat{X}_2}(\bar{v})). \end{aligned}$$

It follows $\widehat{iso}_V(track_{\hat{S} \Rightarrow \hat{U}_1 \Rightarrow^* \hat{X}_1}(\bar{v})) = track_{\hat{S} \Rightarrow \hat{U}_2 \Rightarrow^* \hat{X}_2}(\bar{v})$ as \hat{X}_2^\bullet is fully collapsed. Thus

$$\widehat{iso} \circ tr_1 = tr_2, \tag{7.6}$$

where tr_i is the restriction of $track_{\widehat{S} \Rightarrow \widehat{U}_i \Rightarrow^* \widehat{X}_i}$ to $V_{\widehat{X}_i^\bullet}$ ($i = 1, 2$).

Restricting the derivation $\widehat{S} \Rightarrow_{p_i, \widehat{g}_i} \widehat{U}_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* \widehat{X}_i$ according to Lemma 7.12² yields a derivation $S \Rightarrow_{p_i, g_i} U_i \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* X_i$ such that $X_i = \widehat{X}_i$ and $track_{S \Rightarrow U_i \Rightarrow^* X_i} = track_{\widehat{S} \Rightarrow \widehat{U}_i \Rightarrow^* \widehat{X}_i}$, for $i = 1, 2$. Let $X_i^\bullet = \widehat{X}_i^\bullet$ for $i = 1, 2$. Since \widehat{iso} is a bijection on the variable-edges of \widehat{X}_1^\bullet and \widehat{X}_2^\bullet , there is a restriction $iso: X_1^\bullet \rightarrow X_2^\bullet$ which is again an isomorphism. With $\widehat{iso}_V = iso_V$ and (7.6) follows $iso_V \circ tr_1 = tr_2$. So the critical pair $U_1 \Leftarrow_{p_1, g_1} S \Rightarrow_{p_2, g_2} U_2$ is joinable, concluding the proof. \square

Theorem 7.13 is particularly useful to prove confluence of $\Rightarrow_{\mathcal{R}}$ in cases where $\rightarrow_{\mathcal{R}}$ is non-terminating. For instance, $\Rightarrow_{\mathcal{R}}$ is terminating for the second non-terminating term rewriting system in Example 5.26. As there is only one critical pair over \mathcal{E} which is easily shown to be joinable, $\Rightarrow_{\mathcal{R}}$ is confluent by Theorem 7.13. In contrast, proving $\rightarrow_{\mathcal{R}}$ confluent (in order to apply Corollary 6.22) requires a proof by induction on the term structure. This is because the Critical Pair Lemma for term rewriting (see [Hue80]) cannot guarantee confluence when $\rightarrow_{\mathcal{R}}$ is non-terminating.

From the proof of Theorem 7.13 a test can be derived which—in the presence of termination—decides whether $\Rightarrow_{\mathcal{R}}$ is confluent or not.

Theorem 7.14 *The procedure in Figure 7.4 solves the following problem:*

Instance: *A term rewriting system \mathcal{R} with finitely many rules such that $\Rightarrow_{\mathcal{R}}$ is terminating.*

Question: *Is $\Rightarrow_{\mathcal{R}}$ confluent?*

Proof Since \mathcal{R} has a finite rule set, \mathcal{E} is finite, too. Then the set of critical pairs over \mathcal{E} is also finite and can be effectively constructed.

The procedure in Figure 7.4 terminates since, by Theorem 6.17, $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is terminating. For showing correctness, assume first that the output is “ $\Rightarrow_{\mathcal{R}}$ is not confluent.” Then there are derivations $\widehat{X}_1 \Leftarrow_{\mathcal{E} \cup \mathcal{F}}^* \widehat{S} \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* \widehat{X}_2$ such that \widehat{X}_1 and \widehat{X}_2 are normal forms. Moreover, there is a node v in \widehat{S} such that $term_{\widehat{X}_1}(track_{\widehat{S} \Rightarrow^* \widehat{X}_1}(v)) \neq term_{\widehat{X}_2}(track_{\widehat{S} \Rightarrow^* \widehat{X}_2}(v))$. Corollaries 5.18 and 5.23 ensure that both terms are normal forms of $term_{\widehat{S}}(v)$, so $\rightarrow_{\mathcal{R}}$ is not confluent and, by Theorem 6.19, neither is $\Rightarrow_{\mathcal{R}}$.

Assume now that the procedure asserts confluence of $\Rightarrow_{\mathcal{R}}$. By Theorem 7.13 it suffices to show that all critical pairs over \mathcal{E} are joinable. Consider any critical pair $U_1 \Leftarrow S \Rightarrow U_2$ and its extension $\widehat{U}_1 \Leftarrow \widehat{S} \Rightarrow \widehat{U}_2$. The procedure constructs derivations $\widehat{U}_1 \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* \widehat{X}_1$ and $\widehat{U}_2 \Rightarrow_{\mathcal{E} \cup \mathcal{F}}^* \widehat{X}_2$ such that \widehat{X}_1 and \widehat{X}_2 are normal forms. In the proof of Theorem 7.13 it is shown that there is an isomorphism $\widehat{X}_1^\bullet \rightarrow \widehat{X}_2^\bullet$,

²Note that the jungles occurring in this derivation are variable-collapsed since evaluation and folding steps do not create variables.

where \widehat{X}_1^\bullet and \widehat{X}_2^\bullet are certain subjungles of \widehat{X}_1 and \widehat{X}_2 . In the same proof it is shown that this morphism has a restriction $X_1^\bullet \rightarrow X_2^\bullet$ that makes the critical pair $U_1 \Leftarrow S \Rightarrow U_2$ joinable in the sense of Definition 7.9. \square

```

input: a term rewriting system  $\mathcal{R}$  with finitely many rules such
that  $\Rightarrow_{\mathcal{R}}$  is terminating

begin
  for each critical pair  $U_1 \Leftarrow S \Rightarrow U_2$  do
    extend  $S$  to a variable-collapsed jungle  $\widehat{S}$  by appending
    variable-edges to open nodes;
    construct extended direct derivations  $\widehat{U}_1 \Leftarrow \widehat{S} \Rightarrow \widehat{U}_2$  according
    to the Extension Lemma 6.2;
    apply evaluation and folding rules as long as possible to  $\widehat{U}_1$ 
    and  $\widehat{U}_2$ , obtaining derivations  $\widehat{U}_1 \Rightarrow_{\mathcal{EUF}}^* \widehat{X}_1$  and  $\widehat{U}_2 \Rightarrow_{\mathcal{EUF}}^* \widehat{X}_2$ 
    such that  $\widehat{X}_1$  and  $\widehat{X}_2$  are normal forms;
     $V := V_{\widehat{S}}$ ;
    repeat
      choose some node  $v$  in  $V$ ;
      if  $term_{\widehat{X}_1}(track_{\widehat{S} \Rightarrow \widehat{U}_1 \Rightarrow^* \widehat{X}_1}(v)) = term_{\widehat{X}_2}(track_{\widehat{S} \Rightarrow \widehat{U}_2 \Rightarrow^* \widehat{X}_2}(v))$ 
      then  $V := V - \{v\}$  else return(“ $\Rightarrow_{\mathcal{R}}$  is not confluent”)
    until  $V = \emptyset$ 
  endfor;
  write(“ $\Rightarrow_{\mathcal{R}}$  is confluent”)
end

```

Figure 7.4: Decision procedure for confluence

Chapter 8

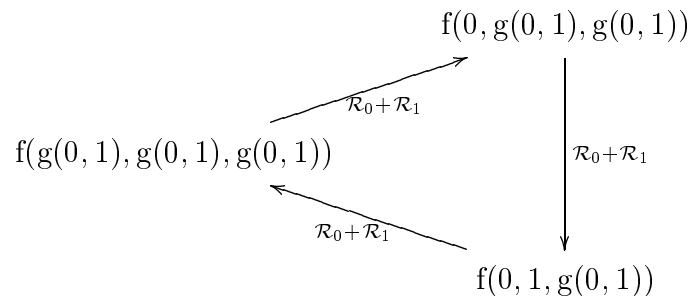
Modularity

Specification and programming languages based on equations usually provide structuring mechanisms to build specifications or programs from smaller pieces (see, for example, the catalogue of implementations in [HKK91] and the references given there). Thus, when specification and program execution consists in term evaluation or when theorem provers are based on rewriting techniques, the question arises whether properties like termination and confluence are modular in the sense that they are preserved by combinations of sets of rewrite rules.

Research in this direction starts with Toyama [Toy87b] who considers the disjoint union $\mathcal{R}_0 + \mathcal{R}_1$ of two term rewriting systems, which is the union of renamed copies of \mathcal{R}_0 and \mathcal{R}_1 that have no common function symbols. He proves that $\mathcal{R}_0 + \mathcal{R}_1$ is confluent if and only if \mathcal{R}_0 and \mathcal{R}_1 are so. But the complexity of the proof indicates that such results are far from obvious. Indeed, Toyama points out that the corresponding statement for termination does not hold [Toy87a]. He gives the following counterexample: The systems

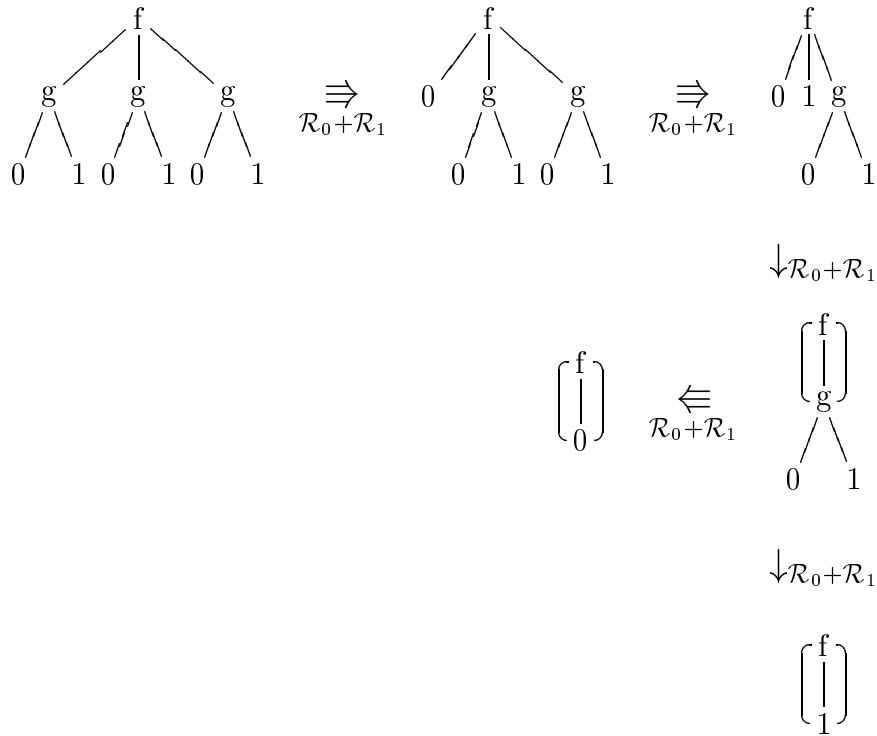
$$\mathcal{R}_0 \left\{ \begin{array}{l} f(0, 1, x) \rightarrow f(x, x, x) \end{array} \right.$$
$$\mathcal{R}_1 \left\{ \begin{array}{l} g(x, y) \rightarrow x \\ g(x, y) \rightarrow y \end{array} \right.$$

are both terminating while $\mathcal{R}_0 + \mathcal{R}_1$ admits the cyclic rewrite sequence shown below.



The undesirable phenomenon revealed by this example stimulated several researchers to look for sufficient conditions on term rewriting systems that guarantee the modularity of termination. For the disjoint union of systems, such conditions are given by Rusinowitch [Rus87], Drosten [Dro89], Middeldorp [Mid89], Toyama, Klop, and Barendregt [TKB89], and Gramlich [Gra92]. Combinations of systems with common function symbols are considered by Bachmair and Dershowitz [BD86, Der81], Geser [Ges90], and Kurihara and Ohuchi [KO92]. Kurihara and Kaji [KK90] establish the termination of a particular rewrite strategy on combined systems. For a comprehensive survey of modularity results for term rewriting systems, the reader may consult Middeldorp's thesis [Mid90].

In the first section of this chapter it is shown that, in contrast to term rewriting, termination of collapsed tree rewriting does behave modular under the disjoint union of systems. In fact, disjointness can be relaxed to so-called "crosswise disjointness" which allows that certain function symbols are shared between the left-hand respectively right-hand sides of the given rule sets. For instance, consider again Toyama's counterexample shown above. Starting with the tree representation of $f(g(0, 1), g(0, 1), g(0, 1))$, collapsed tree rewriting proceeds as follows¹:



The crucial point for the termination of these rewrite sequences is the evaluation step based on the rule $f(0, 1, x) \rightarrow f(x, x, x)$. While term rewriting produces two

¹For space reasons a simplified representation of collapsed trees is used which should be self-explanatory.

copies of the subterm $g(0, 1)$, collapsed tree rewriting yields a shared representation of this subterm. As a result, there are no multiple representations of $g(0, 1)$ which could be evaluated differently and hence there is no chance to apply the evaluation rule for $f(0, 1, x) \rightarrow f(x, x, x)$ again.

In Section 8.2 convergence is shown to be modular for collapsed tree rewriting if, in addition to crosswise disjointness, the left-hand sides of the given rule sets do not mutually overlap. Again, this result does not hold for term rewriting, even if the two systems are disjoint (see Example 8.14).

8.1 Modularity of Termination

The modularity results for termination and convergence of collapsed tree rewriting allow for combinations of term rewriting systems with common sorts and function symbols. Such combinations are formed as componentwise unions of signatures and rule sets.

Definition 8.1 The *union* of two term rewriting systems $\mathcal{R}_0 = \langle \Sigma_0, P_0 \rangle$ and $\mathcal{R}_1 = \langle \Sigma_1, P_1 \rangle$ is the system

$$\mathcal{R}_0 \cup \mathcal{R}_1 = \langle \Sigma, P_0 \cup P_1 \rangle$$

where $\Sigma_V, \Sigma_E, \Sigma_X$ are the unions of the corresponding sets from Σ_0 and Σ_1 , and $Type_{\Sigma}$ is defined on Σ_{iE} like $Type_{\Sigma_i}$, for $i = 0, 1$.

Note that the combined signature Σ is well-defined only if $Type_{\Sigma_0}(f) = Type_{\Sigma_1}(f)$ for each function symbol f in $\Sigma_{0E} \cap \Sigma_{1E}$, which is assumed throughout.

In general, the union of sets of rewrite rules preserves neither termination of term rewriting nor of collapsed tree rewriting. The simplest counterexample is given by the one-rule systems $\{a \rightarrow b\}$ and $\{b \rightarrow a\}$. The sharing of function symbols by left- and right-hand sides characteristic for this example suggests the following definition.

Definition 8.2 (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:

$$\mathcal{R}_0 \begin{cases} f(x) & \rightarrow g(x, x) \\ a & \rightarrow b \end{cases}$$

$$\mathcal{R}_1 \begin{cases} f(f(x)) & \rightarrow g(x, b) \\ h(a, x) & \rightarrow h(b, x) \end{cases}$$

It turns out that the union of crosswise disjoint term rewriting systems preserves termination of collapsed tree rewriting without further restrictions.

Theorem 8.3 (modularity of termination) *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.*

The rest of this section is devoted to the proof of this result. In the following let $\mathcal{R}_0 = \langle \Sigma_0, P_0 \rangle$ and $\mathcal{R}_1 = \langle \Sigma_1, P_1 \rangle$ be two arbitrary crosswise disjoint term rewriting systems. At first some notations are introduced. Let

- Σ be the signature of $\mathcal{R}_0 \cup \mathcal{R}_1$,
- Σ_i^\ominus be the subsignature of Σ_i that is obtained by removing all function symbols that do not occur in any rule of \mathcal{R}_i , for $i = 0, 1$,
- \mathcal{L}_i be the set of all function symbols occurring in the left-hand sides of rules in \mathcal{R}_i , for $i = 0, 1$,
- \mathcal{E}_i and \mathcal{F}_i be the sets of evaluation and folding rules for \mathcal{R}_i ($i = 0, 1$), and $\mathcal{E} = \mathcal{E}_0 \cup \mathcal{E}_1$, $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1$ (so \mathcal{E} and \mathcal{F} consist of the evaluation and folding rules for $\mathcal{R}_0 \cup \mathcal{R}_1$).

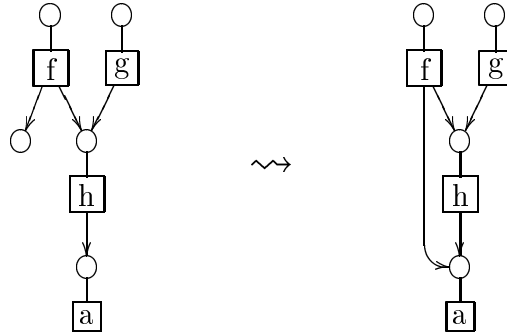
For every Σ -jungle G let

- G^i be the subjungle consisting of all nodes and edges with label in Σ_i^\ominus ,
- $Left(G^i)$ be the subjungle of G^i consisting of all edges with label in \mathcal{L}_i and all source and target nodes of these edges,
- $G^\#$ be the number of edges with label in $\mathcal{L}_0 \cap \mathcal{L}_1$.

To prove Theorem 8.3, a deformation of jungles is considered in which an open node is fused with some other node (which may be also open) such that the resulting hypergraph is again a jungle. Define a relation \rightsquigarrow on Σ -jungles as follows:

$G \rightsquigarrow H$ if H is isomorphic to a jungle obtained from G by identifying an open node with some other node having the same label.

For instance, the following picture shows an \rightsquigarrow -step:



A nice property of \rightsquigarrow is that every relation of evaluation and folding steps can be enriched by \rightsquigarrow -steps without losing termination. In particular, the following property is exploited below.

Lemma 8.4 *If $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i}$ is terminating on closed Σ_i -jungles, then $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i} \cup \rightsquigarrow$ is terminating on arbitrary Σ_i -jungles, for $i = 0, 1$.*

Proof The relation $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i}$ is terminating on arbitrary Σ_i -jungles since, by the Extension Lemma 6.2, all non-closed jungles in a rewrite sequence can be turned into closed jungles by appending variable-labelled edges to open nodes. Moreover, every \rightsquigarrow -step decreases the number of open nodes in a jungle by one while $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i}$ -steps preserve this number. So every sequence of steps over $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i} \cup \rightsquigarrow$ contains only finitely many \rightsquigarrow -steps, and hence $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i} \cup \rightsquigarrow$ is terminating because $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i}$ is. \square

The next two lemmas reveal the basic idea for the proof of Theorem 8.3: In every sequence of $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ -steps the number $G^\#$ remains constant after finitely many steps, and from this point on $\Rightarrow_{\mathcal{E}_i}$ - and $\Rightarrow_{\mathcal{F}}$ -steps affect G^{1-i} —if at all—in a harmless way.

Lemma 8.5 *For arbitrary Σ -jungles G and H ,*

$$G \Rightarrow_{\mathcal{E} \cup \mathcal{F}} H \text{ implies } G^\# \geq H^\#.$$

Proof Let $p = (L \leftarrow K \rightarrow R)$ be the rule applied in the step $G \Rightarrow_{\mathcal{E} \cup \mathcal{F}} H$. Then $H^\# = (G^\# - L^\#) + R^\#$. So it suffices to show $L^\# \geq R^\#$. This holds obviously for folding rules while if p is an evaluation rule, then $L^\# \geq K^\# = R^\#$ since no symbol in $\mathcal{L}_0 \cap \mathcal{L}_1$ occurs in any right-hand side of a rule in $\mathcal{R}_0 \cup \mathcal{R}_1$. \square

Lemma 8.6 *Let $G \Rightarrow_{\mathcal{E}_i \cup \mathcal{F}} H$ ($i \in \{0, 1\}$) be a direct derivation on Σ -jungles such that $G^\# = H^\#$. Then for each subjungle U of G^{1-i} there is a subjungle \bar{U} of H^{1-i} such that*

$$U \cong \bar{U} \text{ or } U \rightsquigarrow \bar{U} \text{ or } U \Rightarrow_{\mathcal{F}_{1-i}} \bar{U}.$$

Proof *Case 1: $G \Rightarrow_{\mathcal{E}_i} H$.* Let e be the edge in $Left(G^i)$ that is removed in the evaluation step. Crosswise disjointness prevents that any edges with label in \mathcal{L}_{1-i} are created. So e is not in $Left(G^{1-i})$, as otherwise $m_G(e) \in \mathcal{L}_0 \cap \mathcal{L}_1$ leads to the contradiction $G^\# > H^\#$. Then e is neither in the rest of G^{1-i} since its label does not occur in any right-hand side of a rule in \mathcal{R}_{1-i} . Therefore the evaluation step can affect the subjungle U at most by the identification of two nodes. That is, there is a subjungle \bar{U} of H^{1-i} that is either isomorphic to U or is obtained by identifying two nodes v_1, v_2 in U . In the latter case either v_1 or v_2 is the source node of e . But this node is an open node in U because e is not in U . Hence $U \rightsquigarrow \bar{U}$.

Case 2: $G \Rightarrow_{\mathcal{F}} H$. By Lemma 4.16 there is a surjective jungle morphism $f: G \rightarrow H$ with $f_V = \text{track}_{G \Rightarrow H}$. Let e_1 and e_2 be the two edges in G such that $e_1 \neq e_2$ and $f_E(e_1) = f_E(e_2)$.

Case 2.1: $\{e_1, e_2\} \subseteq E_U$. Then the applied folding rule is in \mathcal{F}_{1-i} and the Restriction Lemma 6.1 shows that $U \Rightarrow_{\mathcal{F}_{1-i}} fU$.

Case 2.2: $\{e_1, e_2\} \not\subseteq E_U$. Let v_1 and v_2 be the source nodes of e_1 and e_2 .

Case 2.2.1: $\{v_1, v_2\} \subseteq V_U$. Then at least one of the two nodes is an open node in U and hence $U \rightsquigarrow fU$.

Case 2.2.1: $\{v_1, v_2\} \not\subseteq V_U$. Then f is injective on U so that $U \cong fU$. \square

The next lemma provides the last piece for the proof of Theorem 8.3 in that it allows to restrict evaluation steps to particular subjungles.

Lemma 8.7 *Let $G \Rightarrow_{\mathcal{E}_i} H$ be an evaluation step on Σ -jungles and U be a subjungle of G such that $\text{Left}(G^i) \subseteq U \subseteq G^i$. Then there is a direct derivation $U \Rightarrow_{\mathcal{E}_i} W$ such that $\text{Left}(H^i) \subseteq W \subseteq H^i$.*

Proof The image of the left-hand side of the applied evaluation rule lies in U since $\text{Left}(G^i) \subseteq U$. So the Restriction Lemma 6.1 yields a direct derivation $U \Rightarrow_{\mathcal{E}_i} W$ with $W \subseteq H$. Clearly $W \subseteq H^i$ holds as U lies in G^i , and by the proof of the Restriction Lemma it is easy to check that $\text{Left}(H^i) \subseteq W$. \square

Proof of Theorem 8.3 Obviously $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are terminating whenever $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is. Conversely, assume that $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are terminating. Then, by Theorem 6.17, $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i}$ is terminating on closed Σ_i -jungles, for $i = 0, 1$. By the same theorem, it suffices to show that $\Rightarrow_{\mathcal{E} \cup \mathcal{F}}$ is terminating on Σ -jungles. Suppose that there is an infinite sequence

$$G_1 \Rightarrow_{\mathcal{E} \cup \mathcal{F}} G_2 \Rightarrow_{\mathcal{E} \cup \mathcal{F}} G_3 \Rightarrow_{\mathcal{E} \cup \mathcal{F}} \dots$$

Since $\Rightarrow_{\mathcal{F}}$ is terminating, this sequence contains infinitely many $\Rightarrow_{\mathcal{E}_i}$ -steps for some $i \in \{0, 1\}$. Moreover, by Lemma 8.5, the number $G_n^\#$ remains constant after finitely many steps. So there is an sequence

$$H_1 \Rightarrow_{\mathcal{E}_{1-i} \cup \mathcal{F}}^* H_2 \Rightarrow_{\mathcal{E}_i} H_3 \Rightarrow_{\mathcal{E}_{1-i} \cup \mathcal{F}}^* H_4 \Rightarrow_{\mathcal{E}_i} \dots \quad (8.1)$$

such that $H_n^\# = H_{n+1}^\#$ for all $n \geq 1$. Consider any step $H \Rightarrow_{\mathcal{E}_{1-i} \cup \mathcal{F}} \bar{H}$ in (8.1) and any subjungle U of H^i such that $\text{Left}(H^i) \subseteq U \subseteq H^i$. Then, by Lemma 8.6, there is a subjungle \bar{U} of \bar{H}^i such that $U \cong \bar{U}$ or $U \rightsquigarrow \bar{U}$ or $U \Rightarrow_{\mathcal{F}_i} \bar{U}$. Also, $\text{Left}(\bar{H}^i) \subseteq \bar{U}$ holds because the step $H \Rightarrow_{\mathcal{E}_{1-i} \cup \mathcal{F}} \bar{H}$ does not create edges with label in \mathcal{L}_i . Combined with Lemma 8.7, this shows that (8.1) gives rise to a sequence

$$H_1^i = U_1 (\rightsquigarrow \cup \Rightarrow_{\mathcal{F}_i})^* U_2 \Rightarrow_{\mathcal{E}_i} U_3 (\rightsquigarrow \cup \Rightarrow_{\mathcal{F}_i})^* U_4 \Rightarrow_{\mathcal{E}_i} \dots$$

But this contradicts the termination of $\Rightarrow_{\mathcal{E}_i \cup \mathcal{F}_i} \cup \rightsquigarrow$ established in Lemma 8.4, since U_1, U_2, U_3, \dots are Σ_i -jungles. \square

8.2 Modularity of Convergence

Unlike termination, confluence of collapsed tree rewriting is not modular under the disjoint union of systems. In fact, confluence may break down already by signature extensions: $\Rightarrow_{\mathcal{R}}$ is (strongly) confluent for the one-rule system $\{a \rightarrow f(a)\}$ as long as there are only constants and unary function symbols, but an additional binary function symbol causes non-confluence (see Example 5.28). Nevertheless, confluence together with termination turns out to be modular under system combinations that satisfy two conditions. The first condition is crosswise disjointness, guaranteeing the preservation of termination, and the second condition rules out that left-hand sides from different systems overlap.

Definition 8.8 (overlap, non-interference)

- (1) A term l *overlaps* a term l' if there is a non-variable subterm u of l' such that $\sigma(l) = \sigma'(u)$ for some substitutions σ and σ' .
- (2) Two term rewriting systems \mathcal{R}_0 and \mathcal{R}_1 are *non-interfering* if no left-hand side of a rule in \mathcal{R}_0 overlaps the left-hand side of a rule in \mathcal{R}_1 and vice versa.

For example, the term $f(a, x)$ overlaps the term $g(f(x, h(y)))$ by substitutions σ and σ' with $\sigma(x) = h(y)$ and $\sigma'(x) = a$, $\sigma'(y) = y$. Note that for two non-interfering systems \mathcal{R}_0 and \mathcal{R}_1 , left-hand sides of rules within \mathcal{R}_0 and \mathcal{R}_1 may overlap.

Given two evaluation rules p_0 and p_1 over \mathcal{R}_0 respectively \mathcal{R}_1 , non-interference prevents that p_0 and p_1 create a critical pair. This is a consequence of the following lemma.

Lemma 8.9 *Let \mathcal{R}_0 and \mathcal{R}_1 be term rewriting systems with associated sets of evaluation rules \mathcal{E}_0 and \mathcal{E}_1 , and let $p_i = (\underline{L}_i \leftarrow \underline{K}_i \rightarrow \underline{R}_i) \in \mathcal{E}_i$, for $i = 0, 1$. If \mathcal{R}_0 and \mathcal{R}_1 are non-interfering, then each two jungle morphisms $g_0: \underline{L}_0 \rightarrow G$ and $g_1: \underline{L}_1 \rightarrow G$ into a closed jungle G satisfy $g_0 \underline{L}_0 \cap g_1 \underline{L}_1 = g_0 \underline{K}_0 \cap g_1 \underline{K}_1$.*

Proof Let $l_i \rightarrow r_i$ be the rewrite rule underlying p_i , and σ_i be the substitution induced by g_i , for $i = 0, 1$. Suppose that $g_0 \underline{L}_0 \cap g_1 \underline{L}_1 \neq g_0 \underline{K}_0 \cap g_1 \underline{K}_1$. Then, by the construction of evaluation rules, $g_{0E}(e_0)$ is in $g_1 \underline{L}_1$ or $g_{1E}(e_1)$ is in $g_0 \underline{L}_0$, where e_i is the unique edge in $E_{\underline{L}_i} - E_{\underline{K}_i}$ ($i = 0, 1$). Without loss of generality assume the former. Let e'_1 be an edge in \underline{L}_1 such that $g_{1E}(e'_1) = g_{0E}(e_0)$. One

obtains

$$\begin{aligned}
\sigma_0(l_0) &= \sigma_0(\text{term}_{L_0}(\text{root}_{L_0})) \\
&= \text{term}_G(g_{0V}(\text{root}_{L_0})) \quad \text{Lemma 4.10} \\
&= \text{term}_G(g_{0V}(s_{\underline{L}_0}(e_0))) \\
&= \text{term}_G(s_G(g_{0E}(e_0))) \\
&= \text{term}_G(s_G(g_{1E}(e'_1))) \\
&= \text{term}_G(g_{1V}(s_{\underline{L}_1}(e'_1))) \\
&= \sigma_1(\text{term}_{L_1}(s_{\underline{L}_1}(e'_1))). \quad \text{Lemma 4.10}
\end{aligned}$$

But this contradicts the assumed non-interference since $\text{root}_{L_1} >_{L_1} s_{\underline{L}_1}(e'_1)$ implies that $\text{term}_{L_1}(s_{\underline{L}_1}(e'_1))$ is a (non-variable) subterm of $\text{term}_{L_1}(\text{root}_{L_1}) = l_1$. \square

Theorem 3.8 allows to show that the union of non-interfering term rewriting systems over the same signature preserves local confluence of collapsed tree rewriting.

Theorem 8.10 (modularity of local confluence) *Let $\mathcal{R}_0 \cup \mathcal{R}_1$ be the union of two non-interfering term rewriting systems with the same signature. If $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are locally confluent, then so is $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$.*

Proof Let \mathcal{E}_i be the set of evaluation rules for \mathcal{R}_i ($i = 0, 1$). By Theorem 7.5 it suffices to show that for divergent evaluation steps $D_0 \Leftarrow_{\mathcal{E}_0 \cup \mathcal{E}_1} C \Rightarrow_{\mathcal{E}_0 \cup \mathcal{E}_1} D_1$ there is always a collapsed tree X such that $D_0 \Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}^* X \Leftarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}^* D_1$. If $D_0 \Leftarrow_{\mathcal{E}_i} C \Rightarrow_{\mathcal{E}_i} D_1$, then such an X exists since $\Rightarrow_{\mathcal{R}_i}^*$ is locally confluent and $\Rightarrow_{\mathcal{R}_i}^* \subseteq \Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}^*$ ($i = 0, 1$). Therefore the case $D_0 \Leftarrow_{p_0, g_0} C \Rightarrow_{p_1, g_1} D_1$ with $p_0 \in \mathcal{E}_0$ and $p_1 \in \mathcal{E}_1$ remains to be considered. Let M_i be a jungle such that $C \Rightarrow_{p_i, g_i} M_i$ and $M_i / \text{track}_{C \Rightarrow M_i}(\text{root}_C) = D_i$, where $p_i = (L_i \leftarrow K_i \rightarrow R_i)$, for $i = 0, 1$. By Lemma 8.9, non-interference implies $g_0 L_0 \cap g_1 L_1 = g_0 K_0 \cap g_1 K_1$. Hence, by Theorem 3.8, there is a jungle N such that $M_0 \Rightarrow_{p_1} N \Leftarrow_{p_0} M_1$ and $\text{track}_{C \Rightarrow M_0 \Rightarrow N} = \text{track}_{C \Rightarrow M_1 \Rightarrow N}$. Let $v_i = \text{track}_{C \Rightarrow M_i}(\text{root}_C)$ for $i = 0, 1$, so $\text{track}_{M_0 \Rightarrow N}(v_0) = \text{track}_{M_1 \Rightarrow N}(v_1)$. With the Translation Lemma 6.8 follows

$$D_0 = M_0 / v_0 \Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}^\lambda N / \text{track}_{M_0 \Rightarrow N}(v_0) \Leftarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}^\lambda M_1 / v_1 = D_1.$$

\square

It is open whether Theorem 8.10 still holds when the given term rewriting systems have different signatures. By inspecting the proof, this would be the case if signature extensions preserve local confluence. This, however, is not clear since at least confluence may be destroyed.

Nevertheless, the following lemma establishes the preservation of convergence under signature extensions. The proof exploits the fact that signature extensions do not affect confluence of term rewriting.

Lemma 8.11 *Let $\mathcal{R}_0 \cup \mathcal{R}_1$ be the union of two term rewriting systems such that the rule set of \mathcal{R}_1 is empty. If $\Rightarrow_{\mathcal{R}_0}$ is convergent, then so is $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$.*

Proof Theorem 8.3 shows that $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is terminating since $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are crosswise disjoint and terminating ($\Rightarrow_{\mathcal{R}_1}$ consists only of folding steps). Moreover, $\rightarrow_{\mathcal{R}_0}$ is confluent by Theorem 6.19. Then $\rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is also confluent, according to Proposition 3.1.2 in [Mid90]² (which is a special case of Toyama’s Theorem [Toy87b]). With Corollary 6.22 follows the confluence of $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$. \square

Now the main result of this section can be established.

Theorem 8.12 (modularity of convergence) *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 so is $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$.*

Proof By Theorem 8.3, $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is terminating. Hence, by Newman’s Lemma (see page 10), it suffices to show that $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is locally confluent. Let, for $i = 0, 1$, $\mathcal{R}_i = \langle \Sigma_i, P_i \rangle$ and consider $\mathcal{R}'_i = \langle \Sigma, P_i \rangle$ with Σ being the signature of $\mathcal{R}_0 \cup \mathcal{R}_1$. Then $\mathcal{R}'_i = \mathcal{R}_i \cup \langle \Sigma, \emptyset \rangle$ and hence, by Lemma 8.11, $\Rightarrow_{\mathcal{R}'_i}$ is convergent ($i = 0, 1$). Since \mathcal{R}'_0 and \mathcal{R}'_1 are non-interfering and have the same signature, $\Rightarrow_{\mathcal{R}'_0 \cup \mathcal{R}'_1}$ is locally confluent by Theorem 8.10. This is the desired result because $\mathcal{R}_0 \cup \mathcal{R}_1 = \mathcal{R}'_0 \cup \mathcal{R}'_1$. \square

As an immediate consequence of Theorem 8.12, the combination of term rewriting systems with disjoint function symbols preserves convergence of collapsed tree rewriting.

Corollary 8.13 *Let $\mathcal{R}_0 \cup \mathcal{R}_1$ be the union of two term rewriting systems having signatures with disjoint sets of function symbols. If $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are convergent, then so is $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$.*

Proof Disjointness of function symbols clearly implies crosswise disjointness and non-interference, so the proposition follows from Theorem 8.12. \square

Unlike collapsed tree rewriting, term rewriting does not behave modular for convergent systems with disjoint function symbols. That is, Corollary 8.13—let alone Theorem 8.12—does not hold when $\Rightarrow_{\mathcal{R}_0}$, $\Rightarrow_{\mathcal{R}_1}$, and $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ are replaced by the corresponding relations for term rewriting. This is demonstrated by the following counterexample.³

²The proof is for unsorted term rewriting systems but holds also in the many-sorted case.

³Further counterexamples are given by Barendregt and Klop (see [Toy87a]) and Ganzinger and Giegerich [GG87].

Example 8.14 Drosten [Dro89] gives the following two term rewriting systems with disjoint function symbols (presented here in the slightly simplified form of [Mid90]):

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

$$\mathcal{R}_1 \begin{cases} g(x, y, y) & \rightarrow x \\ g(x, x, y) & \rightarrow y \end{cases}$$

Both systems can be shown to be convergent, but the union $\mathcal{R}_0 \cup \mathcal{R}_1$ is non-terminating because of the following cyclic rewrite sequence:

$$\begin{array}{ccc} f(g(0, 1, 1), g(0, 1, 1), g(0, 1, 1)) & \xrightarrow{\mathcal{R}_0 \cup \mathcal{R}_1} & f(0, g(0, 1, 1), g(0, 1, 1)) \\ & \uparrow \mathcal{R}_0 \cup \mathcal{R}_1 & \downarrow \mathcal{R}_0 \cup \mathcal{R}_1 \\ & & f(0, g(2, 1, 1), g(0, 1, 1)) \\ & & \downarrow \mathcal{R}_0 \cup \mathcal{R}_1 \\ f(0, 1, g(0, 1, 1)) & \xleftarrow{\mathcal{R}_0 \cup \mathcal{R}_1} & f(0, g(2, 2, 1), g(0, 1, 1)) \end{array}$$

In contrast, Corollary 8.13 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 6.23. As a consequence, rewriting the collapsed tree $\Delta f(g(0, 1, 1), g(0, 1, 1), g(0, 1, 1))$ as long as possible yields the normal form $\Delta 2$, independently of the chosen order of rewrite steps.

In Theorem 8.12, none of the two requirements imposed on \mathcal{R}_0 and \mathcal{R}_1 can be dropped if convergence of $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ shall be ensured. Without crosswise disjointness, the union of the systems $\{a \rightarrow b\}$ and $\{b \rightarrow a\}$ witnesses that termination may get lost. Omission of non-interference, on the other hand, may cause non-confluence as can be seen from the union of $\{a \rightarrow b\}$ and $\{a \rightarrow c\}$. Moreover, non-interference without crosswise disjointness does not even guarantee confluence. This is shown by the following example, in which the non-terminating system $\{a \rightarrow f(a)\}$ is “refined” to a union of two convergent systems.

Example 8.15 Consider the non-interfering systems

$$\mathcal{R}_0: a \rightarrow f(b)$$

$$\mathcal{R}_1: b \rightarrow a$$

with a common signature that contains a binary function symbol g . Both $\Rightarrow_{\mathcal{R}_0}$ and $\Rightarrow_{\mathcal{R}_1}$ are convergent since $\rightarrow_{\mathcal{R}_0}$ and $\rightarrow_{\mathcal{R}_1}$ are so. But Figure 8.1 demonstrates that $\Rightarrow_{\mathcal{R}_0 \cup \mathcal{R}_1}$ is non-confluent: the two collapsed trees in the lower row have no common reduct.

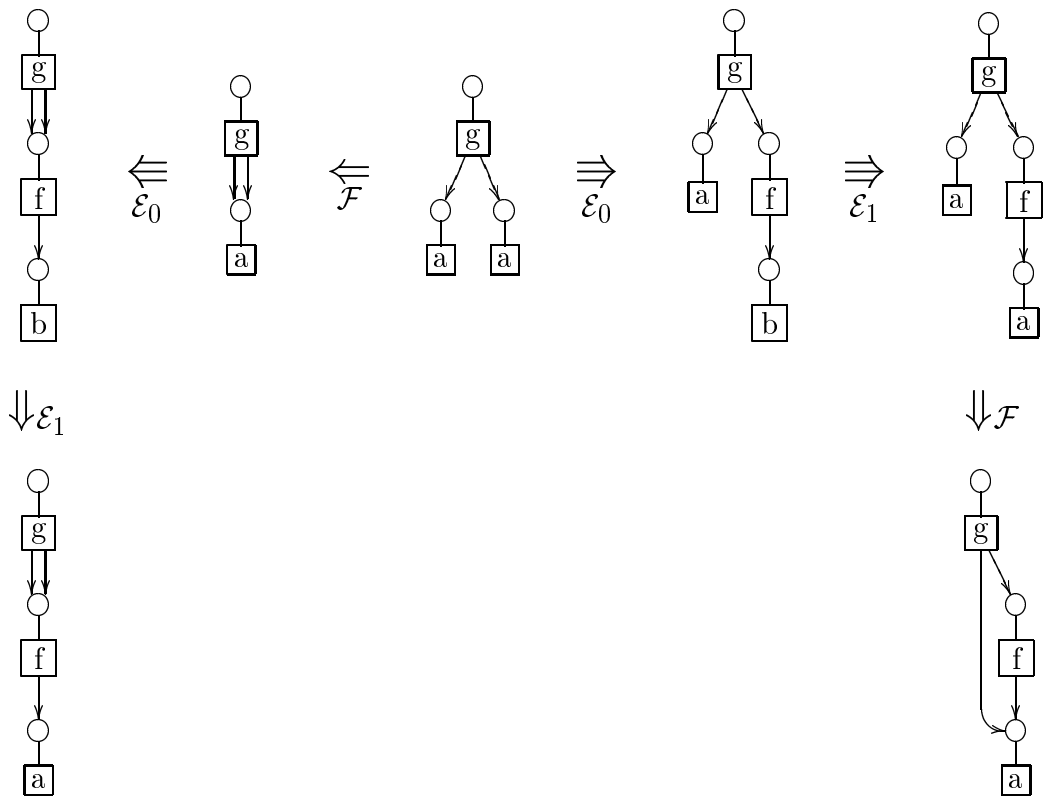


Figure 8.1: Non-confluence of $\equiv_{\mathcal{R}_0 \cup \mathcal{R}_1}$

Chapter 9

Conclusion

9.1 Summary

A hypergraph rewriting approach to the evaluation of functional expressions has been presented, dealing with function definitions in form of directed equations. The outcome is a theory with results about soundness, completeness, termination, confluence, etc., which in particular clarify the relation to the conventional term rewriting approach to expression evaluation.

The proposed approach rests on the “Berlin school” of graph rewriting, using a generalization to the hypergraph case in order to enable a natural representation of expressions. Since rewrite steps in the Berlin approach have only local effects, they cannot include overall garbage collection which is implicit in term rewrite steps.¹ To dispense with garbage collection, however, is unrealistic in view of practical needs and also leads to a cumbersome treatment of completeness and confluence. As a consequence, the Berlin approach is extended by incorporating garbage collection in evaluation steps. The extension is well-behaved in the sense that results for hypergraph rewriting and jungle evaluation can be applied to collapsed tree rewriting via the Translation Lemma 6.8. This lemma provides a smooth transfer of derivations since every rewrite step on a jungle—considered as a collapsed tree augmented with garbage—is localized either to the garbage or to the non-garbage part.

Expressions represented by collapsed trees are evaluated by applying two kind of hypergraph rules, viz. evaluation and folding rules. The former are translated versions of term rewrite rules (i.e. directed equations) while the latter enhance the degree of sharing in collapsed trees without affecting the represented expressions. There is no restriction on the order in which evaluation and folding rules

¹Ehrig, Rosen, and Padawitz [ER80a, Pad82] perform garbage collection by iterated applications of certain removing graph rules which have jungle counterparts in form of the inverses of the “jungle generating rules” in [HKP91]. Such rules together with an appropriate control strategy can be seen as an implementation of the garbage collection introduced in Chapter 6.

are applied, allowing implementations to choose from a wide spectrum of control strategies. The enrichment of the evaluation process by folding steps is a distinctive feature of the present approach, having several advantages:

- Non-left-linear term rewrite rules (i.e. rules with repeated variables in their left-hand sides) do not cause problems as in other graph rewriting approaches to expression evaluation (an exception is [CR93a]).
- Folding can speed up the evaluation process considerably in certain cases. For example, the usual specification of the Fibonacci function by three rewrite rules requires exponential time and space when normal forms are computed by term rewriting. But in [HP91] it is shown that jungle evaluation performs the same task in linear time and space if an appropriate control strategy for evaluation and folding steps is chosen.
- Perhaps the most important consequence of incorporating folding is the achievement of completeness with respect to equational validity. The Completeness Theorem 6.13 reveals that collapsed tree rewriting is a complete mechanism for proving equations, in the same sense as term rewriting is. Other approaches that dispense with folding, such as [BvEG⁺87, CR93a], lack completeness even for left-linear systems (see Example 6.14).

Beside the Completeness Theorem 6.13, the Soundness Theorem 5.14 is a central result of this thesis which shows that evaluation steps on jungles correspond to a certain kind of parallel term rewrite steps. The degree of parallelism depends on the degree of sharing in the rewritten jungles.

Despite the equivalence of collapsed tree rewriting and term rewriting with respect to equational soundness and completeness, the two models differ in their operational behaviour. Collapsed tree rewriting is shown to terminate for more systems than term rewriting does, while the opposite relation holds for confluence. Convergence (i.e. termination and confluence together) is again enjoyed by more systems under collapsed tree rewriting than under term rewriting. As a consequence, the decision procedure for equational validity that rewrites collapsed trees to normal forms and checks equality of the resulting terms (see Figure 6.4 on page 65) is applicable to more systems than the corresponding procedure by term rewriting.

To prove that collapsed tree rewriting is terminating for a given system, the comprehensive stock of techniques developed for term rewriting can be used (see Dershowitz's survey [Der87]). For proving local confluence of collapsed tree rewriting, the Critical Pair Lemma 7.11 provides a sufficient condition which roughly says that all pairs of divergent evaluation steps resulting from overlapping left-hand sides of rules must be confluent in a specific way. Due to the Critical Pair Lemma, a procedure is available that determines whether terminating collapsed tree rewriting is confluent or not (see Figure 7.4 on page 85).

A particularly interesting advantage of collapsed tree rewriting over term rewriting concerns the termination and convergence of combined systems. The modularity results in Chapter 8 show that both properties are preserved under the disjoint union of systems, while term rewriting may become non-terminating in both cases. In fact, the disjointness requirement turns out to be unnecessarily strong and the combined systems are allowed to have certain function symbols in common.

To summarize, expression evaluation by collapsed tree rewriting is sound and complete in the same sense as term rewriting is, but has advantages with respect to efficiency, termination, and modularity.

9.2 Outlook

The research reported in this thesis may be continued in several directions. A challenge coming immediately into mind is the implementation of the proposed approach. Promising first experiences have already been made with the performance of Kahrs's implementations of a functional programming language [Kah92] and of lambda calculus combined with term rewriting [Kah91], in which the idea of combining evaluation and folding has been taken up. In these implementations, a fully collapsed representation of expressions is maintained by storing subgraphs through hashing.

Various parallel modes of applying evaluation and folding rules can be considered to speed up the evaluation of collapsed trees. In the Berlin approach to graph rewriting, concepts and results for parallelism and concurrency have been developed (see for example [EK76, ER80b, KW87]). This knowledge should be profitable for working out a theory of parallel evaluation.

Another extension of collapsed tree rewriting which seems promising is the incorporation of *unification* in order to solve equations by generating substitutions for variables. This aim can be pursued either along the lines of Corradini's and Rossi's approach [CR93a] (see Section 1.4) or by including special hypergraph rules that implement unification steps. The combination of unification and evaluation steps should allow to solve equations in the theory defined by the underlying term rewriting system.

By giving up the acyclicity requirement for jungles, certain infinite terms can be finitely represented and evaluated. Corradini [Cor93] recently developed a theory of infinite parallel term rewriting to provide a semantic basis for the rewriting of cyclic jungles. It will be interesting to consider the topics of this thesis in such a setting.

Finally, one may completely relinquish the jungle structure and consider general hypergraph rewriting as presented in Chapter 3. A first step to study confluence properties in this framework is done in [Plu93b], where a Critical Pair Lemma

is established and confluence of terminating systems is shown to be undecidable. Future research has to find termination criteria for hypergraph rewriting and, in particular, has to prove that properties like confluence and termination are meaningful in various applications.

Appendix A

Basic Mathematical Notions and Notation

FUNCTIONS. The terms “function” and “mapping” are used synonymously. Given a function $f: A \rightarrow B$ and a subset U of A , $f(U)$ is the set $\{f(u) \mid u \in U\}$; an alternative notation is fU . For $b \in B$, the set $\{a \in A \mid f(a) = b\}$ is denoted by $f^{-1}(b)$. The composition $g \circ f: A \rightarrow C$ of two functions $f: A \rightarrow B$ and $g: B \rightarrow C$ is defined by $g \circ f(a) = g(f(a))$.

SETS. $\mathbb{N} = \{0, 1, 2, \dots\}$ is the set of natural numbers. For a finite set A , $|A|$ is the number of elements in A . The power-set of a set A is denoted by $\mathcal{P}(A)$. Given two sets A and B , $A - B$ is the set $\{a \in A \mid a \notin B\}$, while $A + B$ denotes the disjoint union of A and B . No specific construction of $A + B$ is assumed, but the existence of two injective functions $i_A: A \rightarrow A + B$ and $i_B: B \rightarrow A + B$ such that $i_A A \cup i_B B = A + B$ and $i_A A \cap i_B B = \emptyset$. Throughout (except here) the injections are omitted in expressions to enhance readability.

STRINGS. For a set A , A^* is the set of all (finite) strings over A , including the empty string λ . $|w|$ stands for the length of a string w ; for $i = 1, \dots, |w|$, $w|_i$ is the i th element in w . Strings of length 1 are identified with the element that they contain. For an element a in A , $a \in w$ means that a occurs in w . The extension $f^*: A^* \rightarrow B^*$ of a function $f: A \rightarrow B$ maps λ to itself and $a_1 \dots a_n$ to $f(a_1) \dots f(a_n)$.

Appendix B

Proofs for Section 3.2

Proof of Lemma 3.2 Note first that f is well-defined, for if $c(\bar{x}_1) = x = c(\bar{x}_2)$ for some $\bar{x}_1, \bar{x}_2 \in A$, then $b(\bar{x}_1) \sim b(\bar{x}_2)$ and hence $[b(\bar{x}_1)] = [b(\bar{x}_2)]$.

The diagram commutes as for each $x \in A$, $g(b(x)) = [b(x)] = f(c(x))$. To show the universal property, let D' be any set and $f': C \rightarrow D'$, $g': B \rightarrow D'$ be functions such that $g' \circ b = f' \circ c$. Define $h: D \rightarrow D'$ by

$$h(x) = \begin{cases} f'(x) & \text{if } x \in C - cA, \\ g'(\bar{x}) & \text{if } x = [\bar{x}] \in B/\approx. \end{cases}$$

Suppose that h is well-defined. Then for each $x \in B$, $h(g(x)) = h([x]) = g'(x)$, so $h \circ g = g'$. Also, for each $x \in C - cA$, $h(f(x)) = h(x) = f'(x)$ and, for each $x \in cA$, $h(f(x)) = h([b(\bar{x})]) = g'(b(\bar{x})) = f'(c(\bar{x})) = f'(x)$, where $\bar{x} \in A$ with $c(\bar{x}) = x$. Hence $h \circ f = f'$. To see that h is uniquely determined, let $h': D \rightarrow D'$ be any function with $h' \circ f = f'$ and $h' \circ g = g'$. Then for each $[x] \in B/\approx$, $h([x]) = g'(x) = h'(g(x)) = h'([x])$, and for each $x \in C - cA$, $h(x) = h(f(x)) = f'(x) = h'(f(x)) = h'(x)$. Thus $h = h'$.

It remains to show that h is indeed well-defined. That is, if $[x] = [y]$ for some $x, y \in B$, then $g'(x) = g'(y)$ must hold. This is obvious for $x = y$; assume therefore $x \neq y$. Then, by the definition of \approx , there are pairs $\langle a_1, a'_1 \rangle, \dots, \langle a_n, a'_n \rangle$ of elements in A ($n \geq 1$) such that $b(a_1) = x$, $b(a'_n) = y$, $c(a_i) = c(a'_i)$ for $i = 1, \dots, n$, and $b(a'_i) = b(a_{i+1})$ for $i = 1, \dots, n - 1$. With $g' \circ b = f' \circ c$ follows

$$\begin{aligned} g'(x) &= g'(b(a_1)) = f'(c(a_1)) = f'(c(a'_1)) = g'(b(a'_1)) \\ &= g'(b(a_2)) = \dots = g'(b(a'_n)) = g'(y). \end{aligned}$$

□

Proof of Lemma 3.4 Let \approx_V^* be the elementwise extension of \approx_V on V_B^* . The relation \approx_V^* is again an equivalence relation.

Claim. For all $e, e' \in E_B$, $e \approx_E e'$ implies $s_B(e) \approx_V^* s_B(e')$ and $t_B(e) \approx_V^* t_B(e')$.

Proof. The proposition holds obviously if $e = e'$. Let therefore $e \neq e'$. Then there are $e_1, \dots, e_n \in E_B$, $n \geq 2$, such that $e = e_1 \sim_E e_2 \sim_E \dots \sim_E e_n = e'$ (with \sim_E defined as in Construction 3.1). It suffices to consider the case $n = 2$ since the general case follows by transitivity of \approx_V^* . $e_1 \neq e_2$ implies that there are $\bar{e}_1, \bar{e}_2 \in E_A$ with $b_E(\bar{e}_i) = e_i$, for $i = 1, 2$, and $c_E(\bar{e}_1) = c_E(\bar{e}_2)$. Then, by the morphism properties of b and c , $b_V^*(s_A(\bar{e}_i)) = s_B(b_E(\bar{e}_i)) = s_B(e_i)$, for $i = 1, 2$, and $c_V^*(s_A(\bar{e}_1)) = s_C(c_E(\bar{e}_1)) = s_C(c_E(\bar{e}_2)) = c_V^*(s_A(\bar{e}_2))$. Hence $s_B(e_1) \approx_V^* s_B(e_2)$. The equivalence $t_B(e_1) \approx_V^* t_B(e_2)$ is shown analogously. \square

This claim implies that s_D and t_D in Construction 3.3 are well-defined. For if there are $e, e' \in E_B$ with $[e] = [e']$, then $e \approx_E e'$ and hence $s_B(e) \approx_V^* s_B(e')$. It follows $g_V^*(s_B(e)) = g_V^*(s_B(e'))$ by definition of g_V . Analogously one gets $g_V^*(t_B(e)) = g_V^*(t_B(e'))$. Moreover, l_D and m_D are well-defined because equivalent items have the same label. So D is a hypergraph over Σ , provided that $\langle l_D^*(s_D(e)), l_D^*(t_D(e)) \rangle \in \text{Type}(m_D(e))$ for each $e \in E_D$. Consider some $e = [\bar{e}] \in E_B/\approx_E$. Then

$$\begin{aligned} \langle l_D^*(s_D([\bar{e}])), l_D^*(t_D([\bar{e}])) \rangle &= \langle l_D^*(g_V^*(s_B(\bar{e}))), l_D^*(g_V^*(t_B(\bar{e}))) \rangle \\ &= \langle l_D^*([s_B(\bar{e})]^*), l_D^*([t_B(\bar{e})]^*) \rangle \\ &= \langle l_B^*(s_B(\bar{e})), l_B^*(t_B(\bar{e})) \rangle \\ &\in \text{Type}(m_B(\bar{e})) \\ &= \text{Type}(m_D([\bar{e}])). \end{aligned}$$

The case $e \in E_C - c_E E_A$ is handled similarly.

The next task is to show that $f = \langle f_V, f_E \rangle$ and $g = \langle g_V, g_E \rangle$ are hypergraph morphisms. For $e \in E_B$, $s_D(g_E(e)) = s_D([e]) = g_V^*(s_B(e))$ follows immediately from the definition of s_D . Also, for $e \in E_C - c_E E_A$, $s_D(f_E(e)) = s_D(e) = f_V^*(s_C(e))$. For $e \in c_E E_A$,

$$\begin{aligned} s_D(f_E(e)) &= s_D(f_E(c_E(\bar{e}))) \\ &= s_D(g_E(b_E(\bar{e}))) \\ &= s_D([b_E(\bar{e})]) \\ &= g_V^*(s_B(b_E(\bar{e}))) \\ &= g_V^*(b_V^*(s_A(\bar{e}))) \quad b \text{ is morphism} \\ &= f_V^*(c_V^*(s_A(\bar{e}))) \\ &= f_V^*(s_C(c_E(\bar{e}))) \quad c \text{ is morphism} \\ &= f_V^*(s_C(e)), \end{aligned}$$

where $\bar{e} \in E_A$ with $c_E(\bar{e}) = e$. $t_D \circ g_E = g_V^* \circ t_B$ and $t_D \circ f_E = f_V^* \circ t_C$ are shown analogously. Moreover, $l_D \circ g_E = l_B$, $l_D \circ f_E = l_C$, $m_D \circ g_E = m_B$, and $m_D \circ f_E = m_C$ are shown by similar computations.

The diagram in Figure 3.2 clearly commutes, as the corresponding diagrams for node and edge sets are pushouts and hence commutative. So it remains to show the universal property. Let D' be a hypergraph and $g': B \rightarrow D'$, $f': C \rightarrow D'$ be

hypergraph morphisms such that $g' \circ b = f' \circ c$. Then, by the universal properties of the pushouts for node and edge sets, there are unique functions $h_X: X_D \rightarrow X_{D'}$ such that $h_X \circ g_X = g'_X$ and $h_X \circ f_X = f'_X$, for $X = V, E$. The final task is to show that $\langle h_V, h_E \rangle$ is a hypergraph morphism, since uniqueness follows from the uniqueness of h_V and h_E . The equations constituting the morphism property are easily shown by equational reasoning in the style shown above, using case distinctions between items in B/\approx and $C - cA$. \square

Proof of Lemma 3.5 D is well-defined since, by the contact condition, $s_G(e)$ and $t_G(e)$ are in V_D^* for each $e \in E_D$ (note that edges in gK have their source and target nodes in $gK \subseteq D$). Also, d is well-defined because $gK \subseteq D$. By definition, the diagram in Lemma 3.5 is commutative. Let G' together with $h: D \rightarrow G'$ and $g': L \rightarrow G'$ be the pushout of d and $K \rightarrow L$ as defined in Construction 3.3. Then G' has node and edge sets $D/\approx + (L - K)$. By the universal property for pushouts, there is a unique hypergraph morphism $i: G' \rightarrow G$ such that $i(h(x)) = x$ for each $x \in D$ and $i(g'(x)) = g(x)$ for each $x \in L$. It suffices to show that i is an isomorphism. i is surjective since $G = gL \cup D = ig'L \cup ihD$. To show injectivity, let $x_1, x_2 \in G'$ with $i(x_1) = i(x_2)$.

Case 1. $x_1 \in L - K$. Then $g(x_1) = i(g'(x_1)) = i(x_1) = i(x_2)$. Suppose that $x_2 \in D/\approx$. Then $x_2 = h(\bar{x}_2)$ for some $\bar{x}_2 \in D$ and $g(x_1) = i(x_2) = i(h(\bar{x}_2)) = \bar{x}_2$. With the identification condition follows $\bar{x}_2 \in gL - gK$, contradicting the definition of D . Thus $x_2 \in L - K$. Then $g(x_1) = i(x_2) = i(g'(x_2)) = g(x_2)$ and hence $x_1 = x_2$ by the identification condition.

Case 2. $x_1 \in D/\approx$. Then, analogously to case 1, $x_2 \in L - K$ leads to a contradiction. So there are $\bar{x}_1, \bar{x}_2 \in D$ with $h(\bar{x}_i) = x_i$ for $i = 1, 2$. Then $\bar{x}_1 = i(h(\bar{x}_1)) = i(x_1) = i(x_2) = i(h(\bar{x}_2)) = \bar{x}_2$ and consequently $x_1 = x_2$. \square

Bibliography

- [AHS90] Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and Concrete Categories*. Wiley, 1990.
- [Bac91] Leo Bachmair. *Canonical Equational Proofs*. Birkhäuser, 1991.
- [BD86] Leo Bachmair and Nachum Dershowitz. Commutation, transformation, and termination. In *Proc. Automated Deduction*, pages 5–20. Springer Lecture Notes in Computer Science 230, 1986.
- [Bir35] Garrett Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.
- [BO93] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Texts and Monographs in Computer Science. Springer-Verlag, 1993.
- [BvEG⁺87] Hendrik Barendregt, Marko van Eekelen, John Glauert, Richard Kennaway, Rinus Plasmeijer, and Ronan Sleep. Term graph rewriting. In *Proc. Parallel Architectures and Languages Europe*, pages 141–158. Springer Lecture Notes in Computer Science 259, 1987.
- [CG91] Pierre-Louis Curien and Giorgio Ghelli. On confluence for weakly normalizing systems. In *Proc. Rewriting Techniques and Applications*, pages 215–225. Springer Lecture Notes in Computer Science 488, 1991.
- [CMR⁺91] 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*, pages 221–237. Springer Lecture Notes in Computer Science 532, 1991.
- [Coh81] Paul M. Cohn. *Universal Algebra*. D. Reidel, 1981.
- [Cor93] Andrea Corradini. Term rewriting in CT_{Σ} . In *Proc. TAPSOFT '93*. Springer Lecture Notes in Computer Science, 1993. To appear.

- [Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, chapter 5. Elsevier, 1990.
- [CR93a] Andrea Corradini and Francesca Rossi. Hyperedge replacement jungle rewriting for term rewriting systems and logic programming. *Theoretical Computer Science*, 109:7–48, 1993.
- [CR93b] Andrea Corradini and Francesca Rossi. A new term graph rewriting formalism: Hyperedge replacement jungle rewriting. In Roman Sleep, Rinus Plasmeijer, and Marko van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 8. John Wiley, 1993.
- [CRPP91] Andrea Corradini, Francesca Rossi, and Francesco Parisi-Presicce. Logic programming as hypergraph rewriting. In *Proc. CAAP '91*, pages 275–295. Springer Lecture Notes in Computer Science 493, 1991.
- [Der81] Nachum Dershowitz. Termination of linear rewriting systems (preliminary version). In *Proc. Automata, Languages, and Programming*, pages 448–458. Springer Lecture Notes in Computer Science 115, 1981.
- [Der87] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.
- [DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6. Elsevier, 1990.
- [DM79] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.
- [Dro89] Klaus Drost. *Termersetzungssysteme*. Informatik-Fachberichte 210. Springer-Verlag, 1989.
- [EHKPP91] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Mathematical Structures in Computer Science*, 1:361–404, 1991.
- [Ehr77] Hartmut Ehrig. Embedding theorems in the algebraic theory of graph-grammars. In *Proc. Fundamentals of Computation Theory*, pages 245–255. Springer Lecture Notes in Computer Science 56, 1977.

- [Ehr79] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In *Proc. Graph-Grammars and Their Application to Computer Science and Biology*, pages 1–69. Springer Lecture Notes in Computer Science 73, 1979.
- [EK76] Hartmut Ehrig and Hans-Jörg Kreowski. Parallelism of manipulations in multidimensional information structures. In *Proc. Mathematical Foundations of Computer Science*, pages 284–293. Springer Lecture Notes in Computer Science 45, 1976.
- [EK79] Hartmut Ehrig and Hans-Jörg Kreowski. Pushout-properties: An analysis of gluing constructions for graphs. *Mathematische Nachrichten*, 91:135–149, 1979.
- [EKR91] Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Proc. Graph Grammars and Their Application to Computer Science*. Springer Lecture Notes in Computer Science 532, 1991.
- [EM85] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1*. Springer-Verlag, 1985.
- [ENR83] Hartmut Ehrig, Manfred Nagl, and Grzegorz Rozenberg, editors. *Proc. Graph-Grammars and Their Application to Computer Science*. Springer Lecture Notes in Computer Science 153, 1983.
- [ENRR87] Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg, and Azriel Rosenfeld, editors. *Proc. Graph-Grammars and Their Application to Computer Science*. Springer Lecture Notes in Computer Science 291, 1987.
- [ER76] 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.
- [ER80a] Hartmut Ehrig and Barry K. Rosen. The mathematics of record handling. *SIAM Journal on Computing*, 9(3):441–469, 1980.
- [ER80b] Hartmut Ehrig and Barry K. Rosen. Parallelism and concurrency of graph manipulations. *Theoretical Computer Science*, 11:247–275, 1980.
- [FGKM85] Kokichi Futatsugi, Joseph Goguen, Claude Kirchner, and José Meseguer. Principles of OBJ2. In *Proc. 12th ACM Symposium on Principles of Programming Languages*, pages 52–66. ACM, 1985.

- [FK87] Joseph H. Fasel and Robert M. Keller, editors. *Graph Reduction*. Springer Lecture Notes in Computer Science 279, 1987.
- [FW90] William M. Farmer and Ronald J. Watro. Redex capturing in term graph rewriting. *International Journal on Foundations of Computer Science*, 1(4), 1990.
- [FW91] William M. Farmer and Ronald J. Watro. Redex capturing in term graph rewriting. In *Proc. Rewriting Techniques and Applications*, pages 13–24. Springer Lecture Notes in Computer Science 488, 1991.
- [GAL92] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Proc. 19th Annual ACM Symposium on Principles of Programming Languages*, pages 15–26. ACM Press, 1992.
- [Ges90] Alfons Geser. Relative termination. Dissertation, Fakultät für Mathematik und Informatik, Universität Passau, 1990. Available as report no. 91-03, Fakultät für Informatik, Universität Ulm, 1991.
- [GG87] Harald Ganzinger and Robert Giegerich. A note on termination in combinations of heterogeneous term rewriting systems. *Bulletin of the European Association for Theoretical Computer Science*, 31:22–28, 1987.
- [GH86] Alfons Geser and Heinrich Hussmann. Experiences with the RAP system: A specification interpreter combining term rewriting and resolution. In *Proc. ESOP '86*, pages 339–350. Springer Lecture Notes in Computer Science 213, 1986.
- [GKM87] Joseph Goguen, Claude Kirchner, and José Meseguer. Concurrent term rewriting as a model of computation. In *Proc. Graph Reduction*, pages 53–93. Springer Lecture Notes in Computer Science 279, 1987.
- [Gra92] Bernhard Gramlich. Generalized sufficient conditions for modular termination of rewriting. In *Proc. Algebraic and Logic Programming*, pages 53–68. Springer Lecture Notes in Computer Science 632, 1992.
- [Hab89] Annegret Habel. Hyperedge replacement: Grammars and languages. Dissertation, Universität Bremen, Fachbereich Mathematik und Informatik, 1989. Revised version as volume 643 of Springer Lecture Notes in Computer Science, 1992.
- [Hab92] Annegret Habel. Hypergraph grammars: Transformational and algorithmic aspects. *Journal of Information Processing and Cybernetics*, 5:241–277, 1992.

- [HKK91] Miki Hermann, Claude Kirchner, and Hélène Kirchner. Implementations of term rewriting systems. *The Computer Journal*, 34(1):20–33, 1991.
- [HKP88] Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump. Jungle evaluation. In *Proc. Recent Trends in Data Type Specification*, pages 92–112. Springer Lecture Notes in Computer Science 332, 1988.
- [HKP91] Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump. Jungle evaluation. *Fundamenta Informaticae*, 15(1):37–60, 1991.
- [HO80] 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.
- [Hof83] Berthold Hoffmann. Compiler generation: From language definitions to abstract compilers. Dissertation, Technische Universität Berlin, Fachbereich Informatik, 1983.
- [Hof92] Berthold Hoffmann. Term rewriting with sharing and memoization. In *Proc. Algebraic and Logic Programming*, pages 128–142. Springer Lecture Notes in Computer Science 632, 1992.
- [Hol91] Ian Holyer. *Functional Programming with Miranda*. Pitman, 1991.
- [HP88a] 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.
- [HP88b] Berthold Hoffmann and Detlef Plump. Jungle evaluation for efficient term rewriting. Technical Report 4/88, Universität Bremen, Fachbereich Mathematik und Informatik, 1988.
- [HP91] Berthold Hoffmann and Detlef Plump. Implementing term rewriting by jungle evaluation. *RAIRO Theoretical Informatics and Applications*, 25(5):445–472, 1991.
- [Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
- [Hug89] John Hughes. Why functional programming matters. *The Computer Journal*, 32(2):98–107, 1989.

- [Jan88] Matthias Jantzen. *Confluent String Rewriting*, volume 14 of *EATCS Monographs*. Springer-Verlag, 1988.
- [Kah91] Stefan Kahrs. λ -rewriting. Dissertation, Universität Bremen, Fachbereich Mathematik und Informatik, 1991.
- [Kah92] Stefan Kahrs. Unlimp: Uniqueness as a leitmotiv for implementation. In *Proc. Programming Language Implementation and Logic Programming*. Springer Lecture Notes in Computer Science 631, 1992.
- [Kat90] Vinod Kumar Kathail. Optimal interpreters for lambda-calculus based functional languages. PhD thesis, Massachusetts Institute of Technology, 1990.
- [KB70] 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.
- [KK90] Masahito Kurihara and Ikuo Kaji. Modular term rewriting systems and the termination. *Information Processing Letters*, 34:1–4, 1990.
- [KKSdV93] Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. The adequacy of term graph rewriting for simulating term rewriting. In Ronan Sleep, Rinus Plasmeijer, and Marko van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 12. John Wiley, 1993.
- [Klo92] 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.
- [KO92] Masahito Kurihara and Azuma Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. *Theoretical Computer Science*, 103:273–282, 1992.
- [Kre77] Hans-Jörg Kreowski. Manipulationen von Graphmanipulationen. Dissertation, Technische Universität Berlin, 1977.
- [KV90] Claude Kirchner and Patrick Viry. Implementing parallel rewriting. In *Proc. Programming Language Implementation and Logic Programming*, pages 1–15. Springer Lecture Notes in Computer Science 456, 1990.

- [KW87] Hans-Jörg Kreowski and Anne Wilharm. Is parallelism already concurrency? Part 2: Non-sequential processes in graph grammars. In *Proc. Graph-Grammars and Their Application to Computer Science*, pages 361–377. Springer Lecture Notes in Computer Science 291, 1987.
- [Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In *Proc. 17th Annual ACM Symposium on Principles of Programming Languages*, pages 16–30. ACM Press, 1990.
- [LE91] Michael Löwe and Hartmut Ehrig. Algebraic approach to graph transformation based on single pushout derivations. In *Proc. Graph-Theoretic Concepts in Computer Science*, pages 338–353. Springer Lecture Notes in Computer Science 484, 1991.
- [Löw90] Michael Löwe. Implementing algebraic specifications by graph transformation systems. *Journal on Information Processing and Cybernetics (EIK)*, 26(11/12):615–641, 1990.
- [Löw93] Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.
- [MH92] 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.
- [Mid89] Aart Middeldorp. A sufficient condition for the termination of the direct sum of term rewriting systems. In *Proc. Symposium on Logic in Computer Science*, pages 396–401. IEEE Computer Society Press, 1989.
- [Mid90] Aart Middeldorp. Modular properties of term rewriting systems. Dissertation, Vrije Universiteit, Amsterdam, 1990.
- [New42] M.H.A. Newman. On theories with a combinatorial definition of “equivalence”. *Annals of Mathematics*, 43(2):223–243, 1942.
- [O’D85] Michael J. O’Donnell. *Equational Logic as a Programming Language*. MIT Press, 1985.
- [Pad82] Peter Padawitz. Graph grammars and operational semantics. *Theoretical Computer Science*, 19:117–141, 1982.
- [Pau91] Laurence C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.

- [Pey87] Simon L. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice-Hall, 1987.
- [Plu86] Detlef Plump. Im Dschungel: Ein neuer Graph-Grammatik-Ansatz zur effizienten Auswertung rekursiv definierter Funktionen. Diploma thesis, Universität Bremen, Fachbereich Mathematik und Informatik, 1986.
- [Plu91a] Detlef Plump. Graph-reducible term rewriting systems. In *Proc. Graph Grammars and Their Application to Computer Science*, pages 622–636. Springer Lecture Notes in Computer Science 532, 1991.
- [Plu91b] Detlef Plump. Implementing term rewriting by graph reduction: Termination of combined systems. In *Proc. Conditional and Typed Rewriting Systems*, pages 307–317. Springer Lecture Notes in Computer Science 516, 1991.
- [Plu93a] Detlef Plump. Collapsed tree rewriting: Completeness, confluence, and modularity. In *Proc. Conditional Term Rewriting Systems*, pages 97–112. Springer Lecture Notes in Computer Science 656, 1993.
- [Plu93b] Detlef Plump. Hypergraph rewriting: Critical pairs and undecidability of confluence. In Ronan Sleep, Rinus Plasmeijer, and Marko van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 15. John Wiley, 1993.
- [PPEM87] Francesco Parisi-Presicce, Hartmut Ehrig, and Ugo Montanari. Graph rewriting with unification and composition. In *Proc. Graph Grammars and Their Application to Computer Science*. Springer Lecture Notes in Computer Science 291, 1987.
- [RB88] David E. Rydeheard and Rod M. Burstall. *Computational Category Theory*. Prentice Hall, 1988.
- [Ros73] Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973.
- [Ros75a] Barry K. Rosen. A Church-Rosser theorem for graph grammars (announcement). *SIGACT News*, 7(3):26–31, 1975.
- [Ros75b] Barry K. Rosen. Deriving graphs from graphs by applying a production. *Acta Informatica*, 4:337–357, 1975.
- [Rus87] Michaël Rusinowitch. On termination of the direct sum of term rewriting systems. *Information Processing Letters*, 26:65–70, 1987.

- [SA91] Volker Sperschneider and Grigorios Antoniou. *Logic: A Foundation for Computer Science*. Addison-Wesley, 1991.
- [SPvE93] Ronan Sleep, Rinus Plasmeijer, and Marko van Eekelen, editors. *Term Graph Rewriting: Theory and Practice*. John Wiley, 1993.
- [Sta79] John Staples. A graph-like lambda calculus for which leftmost-outermost reduction is optimal. In *Proc. Graph-Grammars and Their Application to Computer Science*, pages 440–454. Springer Lecture Notes in Computer Science 73, 1979.
- [Sta80a] John Staples. Computation on graph-like expressions. *Theoretical Computer Science*, 10:171–185, 1980.
- [Sta80b] John Staples. Optimal evaluations of graph-like expressions. *Theoretical Computer Science*, 10:297–316, 1980.
- [Sta80c] John Staples. Speeding up subtree replacement systems. *Theoretical Computer Science*, 11:39–47, 1980.
- [Sta83] John Staples. Two level representation for faster evaluation. In *Proc. Graph-Grammars and Their Application to Computer Science*, pages 392–404. Springer Lecture Notes in Computer Science 153, 1983.
- [TKB89] Yoshihito Toyama, Jan Willem Klop, and Hendrik Barendregt. Termination for the direct sum of left-linear term rewriting systems. In *Proc. Rewriting Techniques and Applications*, pages 477–491. Springer Lecture Notes in Computer Science 355, 1989.
- [Toy87a] Yoshihito Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
- [Toy87b] Yoshihito Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
- [Tur79] David A. Turner. A new implementation technique for applicative languages. *Software—Practise and Experience*, 9(1):31–49, 1979.
- [Vir92] Patrick Viry. La réécriture concurrente. Thèse de doctorat, Université de Nancy I, 1992.
- [Wad71] Christopher P. Wadsworth. Semantics and pragmatics of the lambda-calculus. PhD thesis, University of Oxford, 1971.

- [Wir90] Martin Wirsing. Algebraic specification. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 13. Elsevier, 1990.
- [WS73] S.A. Walker and H.R. Strong. Characterizations of flowchartable recursions. *Journal of Computer and System Sciences*, 7:404–447, 1973.

Index

- $\langle A, \rightarrow \rangle$, 8
- \rightarrow , 8
- \rightarrow^λ , 8
- \rightarrow^+ , 8
- \rightarrow^* , 8
- \leftarrow , 8
- \leftrightarrow , 8
- \leftrightarrow^* , 8
- $\rightarrow_{\mathcal{R}}$, 31
- $\langle \mathcal{H}, \Rightarrow_{\mathcal{S}} \rangle$, 18
- \Rightarrow_p , 17
- $\Rightarrow_{p,g}$, 18
- $\Rightarrow_{\mathcal{S}}$, 18
- $\Rightarrow_{\mathcal{S}}^\lambda$, 18
- $\Rightarrow_{\mathcal{S}}^*$, 18
- $\Rightarrow_{\mathcal{F}}$, 27
- $\Rightarrow_{\mathcal{E}}$, 34
- $\langle \mathfrak{J}, \Rightarrow_{\mathcal{E} \cup \mathcal{F}} \rangle$, 45
- $\langle \mathfrak{C}, \Rightarrow_{\mathcal{R}} \rangle$, 59
- $\Rightarrow_{\mathcal{E}}$, 58
- $\Rightarrow_{\mathcal{R}}$, 58
- \rightsquigarrow , 89
- $>_G$, 13
- \geq_G , 13
- \subseteq , 13
- \cong , 13
- Δt , 61
- $\blacktriangle t$, 61
- E , 12
- G/v , 37
- G^i , 89
- $G^\#$, 89
- $T(\Sigma)$, 22
- $T_s(\Sigma)$, 22
- V , 12
- $\diamond t$, 33
- \mathcal{E} , 34
- \mathcal{F} , 26
- \mathcal{L}_i , 89
- $Left(G^i)$, 89
- $Mod(E)$, 31
- $PATH_G$, 72
- $Pos(t)$, 37
- \mathcal{R} , 31
- $ROOT_G$, 50
- Σ , 12
- Σ -algebra, 30
- Σ_E , 12
- Σ_V , 12
- \doteq , 30
- \mathfrak{C} , 59
- \mathfrak{J} , 45
- \gg , 46–48
- \mathcal{H} , 18
- \models , 30, 31
- \Rightarrow_{Δ} , 38
- Σ_i^\ominus , 89
- \succ , 46–48
- $TERM_G$, 23
- $Type_\Sigma$, 12
- \underline{G} , 23
- $indegree_G$, 13
- l , 12
- m , 12
- $node_{G/v}$, 37
- occ , 13
- $outdegree_G$, 13
- $root_T$, 33

- s , 12
- t , 12
- t/π , 37
- $term_G$, 23
- $track$, 19

- abstract reduction system, 8
- acyclic hypergraph, 13
- algebra
 - Σ -, 30
- algebraic specification, 2, 6
- applicability
 - of evaluation rules, 41
- argument sort, 21

- Berlin approach, 6
- bottom-up induction, 22

- Church-Rosser property, 9
- closed jungle, 21
- closure
 - of a relation, 8
 - reflexive, 8
 - symmetric, 8
 - symmetric-transitive-reflexive, 8
 - transitive, 8
 - transitive-reflexive, 8
 - under isomorphism, 18
- codomain, 13
- collapsed tree, 57
 - at a node, 37
- collapsed tree at a node, 37
- collapsed tree rewriting, 53, 58
 - completeness of, 60, 63
 - confluence of, 68, 72
 - convergence of, 71
 - local confluence of, 72, 76
 - soundness of, 58
 - termination of, 59
- combination
 - of term rewriting systems, 86, 88
- common reduct, 8
- commutativity of pushout, 14

- completely folded tree, 61
- completeness
 - of collapsed tree rewriting, 60, 63
 - of term rewriting, 31
- Completeness Theorem, 63
- composition
 - of hypergraph morphisms, 13
 - of pushouts, 17
- confluence, 9
 - decision procedure for, 84–85
 - local, 9, *see* local confluence
 - of collapsed tree rewriting, 68, 72
 - of folding, 29
 - of hypergraph rewriting, 99
 - of jungle evaluation, 49
 - of term rewriting, 68, 69
 - strong, 9
 - up to garbage, 50–52
- constant, 21
- contact condition, 16
- context, 31
- convergence, 9
 - modularity of, 92
 - of collapsed tree rewriting, 71
 - of term rewriting, 71
- convertible, 8
- critical pair, 72, 78
 - joinable, 79
- Critical Pair Lemma, 80
- crosswise disjointness, 88
- cyclic graphs, 6, 99

- dag, 20
- decision procedure
 - for confluence, 84–85
 - for validity in $Mod(\mathcal{R})$, 65
- derivation, 18
 - direct, 17–18
 - construction of, 18
 - extension of, 55
 - restriction of, 54
- directed acyclic graph, 20

- domain, 13
- double-pushout approach, 11
- edge, 12
- equation, 30
 - valid, 30
- equational validity, *see* validity in $Mod(\mathcal{R})$
- evaluation
 - expression, 1–3
 - jungle, 30
- evaluation rule, 33
- evaluation step, 34
- expression evaluation, 1–3, 5–7
- extension
 - of $\Rightarrow_{\mathcal{R}}$ -step, 65
 - of derivation, 55
- Extension Lemma, 55
- folding, 27
 - confluence of, 29
 - existence of, 61
- folding rule, 26–27
- folding step, 27
- fully collapsed jungle, 25
 - characterization of, 28
 - uniqueness of, 26
- function symbol, 21
- functional programming, 1–2, 7, 99
- garbage, 57
 - confluence up to, 50–52
 - incompleteness through, 53
 - non-confluence through, 49
- garbage collection, 53–54, 57
- gluing condition, 16
- graph reduction, 7
- high-level replacement system, 11
- hyperedge, 12
- hypergraph, 12
 - acyclic, 13
 - picture of, 12
- hypergraph morphism, 13
 - injective, 13
 - restriction of, 13
 - surjective, 13
- hypergraph pushout, 13
 - construction of, 15
- hypergraph rule, 17
- identification condition, 16
- incident, 13
- inclusion, 13
- incompleteness
 - of $\Leftrightarrow_{\mathcal{R}}^*$, 63
 - through garbage, 53
- induced subhypergraph, 13
- induced substitution, 24
- induction
 - bottom-up, 22
 - top-down, 22
- infix notation, 34
- interface (of a rule), 17
- isomorphic, 13
- isomorphism, 13
- joinability
 - of critical pair, 79
- jungle, 21
 - closed, 21
 - cyclic, 99
 - fully collapsed, 25
 - characterization of, 28
 - uniqueness of, 26
 - variable-collapsed, 24
- jungle evaluation, 30
 - confluence of, 49
 - termination of, 45
- jungle morphism, 23
- label, 12
- lambda calculus, 5, 7, 99
- left-hand side (of a rule), 17
- lifting
 - of term rewrite step, 62
- Lifting Lemma, 62

- local confluence, 9
 - modularity of, 93
 - of $\Rightarrow_{\varepsilon}$, 77
 - of collapsed tree rewriting, 69, 72, 76
 - of term rewriting, 69
- logic programming, 6
- memoization, 6
- model, 31
- modularity, 86
 - of convergence, 92
 - of local confluence, 93
 - of termination, 86, 88
- morphism
 - and substitution, 42
 - hypergraph, 13
 - jungle, 23
- multiset, 46
- Newman's Lemma, 10
- node, 12
 - open, 21
 - source, 12
 - target, 12
- non-interference, 92
- normal form, 9, 45
- normalization, 9
 - of jungle evaluation, 48
- notation
 - infix, 34
- open node, 21
- optimal reduction, 5
- overlap, 92
- path, 40
- position (in a term), 37
- predecessor, 13
- pushout, 13
 - commutativity of, 14
 - properties of, 16
 - set, 14
 - universal property of, 14
- pushout complement, 15
- reachable, 13
- relation
 - term rewrite, 31
- removal of variables, 82
- restriction
 - of a hypergraph morphism, 13
 - of derivation, 54
- Restriction Lemma, 54
- result sort, 21
- right-hand side (of a rule), 17
- roots, 50
- rule
 - application of, 18
 - evaluation, 33
 - folding, 26–27
 - hypergraph, 17
 - interface of, 17
 - left-hand side of, 17
 - right-hand side of, 17
 - term rewrite, 31
- set pushout, 14
 - construction of, 14
- sharing
 - non-confluence through, 49
- signature, 12
- single-pushout approach, 11
- sort, 21
 - argument, 21
 - result, 21
- soundness
 - of collapsed tree rewriting, 58
 - of jungle evaluation, 38
 - of term rewriting, 31
- Soundness Theorem, 38
- source node, 12
- source string, 12
- strong confluence, 9
- subhypergraph, 13

- induced, 13
- subjungle, 23
- substitution, 24
 - and morphism, 42
 - induced, 24
- subterm, 22
- target node, 12
- target string, 12
- term, 22
- term graph rewriting, 5
- term representation, 23
- term rewrite relation, 31
- term rewrite rule, 31
- term rewriting
 - confluence of, 68
 - convergence of, 71
 - unique normal forms of, 68
- term rewriting system, 31
- termination, 9
 - modularity of, 86, 88
 - of collapsed tree rewriting, 59, 65
 - of jungle evaluation, 45, 65
- top-down induction, 22
- Toyama's counterexample, 86–88
- track function, 18–19
- Translation Lemma, 59
- tree, 61
 - completely folded, 61
 - with shared variables, 33
- tree rewriting, 3
- type (of a hyperedge label), 12
- typed hypergraph, 12
- unification, 99
- union
 - of term rewriting systems, 88
- unique normal forms, 10
 - of collapsed tree rewriting, 68
 - of term rewriting, 68
- universal property of pushout, 14
- valid equation, 30
- validity in $Mod(\mathcal{R})$, 31, 60
 - decision procedure for, 65
- valuation, 30
- variable, 21
 - removal of, 82
- variable-collapsed jungle, 24
- vertex, 12