TERESE LITE

Excerpts from the book
TERM REWRITING SYSTEMS *by* Terese

Contents

Preface **ix**

0 Introduction **1**
0.1 An example from functional programming . . . 3
0.2 An example from topology: Reidemeister moves . . . 3
0.3 An example from logic: tautology checking . . . 4

1 Abstract reduction systems **7**
Marc Bezem, Jan Willem Klop
1.1 Basic notions of ARSs . . . 7
1.2 Interrelations between ARS properties . . . 14
1.3 Specific ARS properties . . . 18
1.3.1 Confluence . . . 18
1.3.2 Normalization . . . 19
1.3.3 Miscellaneous . . . 20

2 First-order term rewriting systems **21**
Jan Willem Klop, Roel de Vrijer
2.1 Terms . . . 22
2.1.1 Contexts, occurrences . . . 23
2.1.2 Term trees . . . 25
2.1.3 Substitution . . . 27
2.1.4 Matching, subsumption, unification . . . 29
2.2 Definition of a TRS . . . 31
2.3 Reduction . . . 34
2.4 Equational systems . . . 37
2.5 Semantics . . . 40
2.6 Complete term rewriting systems and the word problem . . . 42
2.7 Overlap and critical pairs . . . 44
2.7.1 Overlapping redex patterns . . . 44
2.7.2 Critical pairs and the Critical Pair Lemma . . . 48
2.7.3 Roads to confluence . . . 53

3 Examples of TRSs and special rewriting formats **55**
Jan Willem Klop, Roel de Vrijer
3.1 Numerals and streams . . . 55
3.2 Combinatory logic . . . 57
3.2.1 The combinators . . . 59
3.2.2 Miscellaneous facts about combinators . . . 62

3.2.3 Fixed points . . . 63
3.2.4 Mixed notation . . . 65
3.2.5 Currying . . . 66
3.3 Special rewriting formats . . . 67
3.3.1 Ground TRSs . . . 67
3.3.2 Constructor term rewriting systems . . . 69
3.3.3 Many-sorted term rewriting systems . . . 70
3.3.4 String rewriting systems . . . 70
3.4 Conditional term rewriting systems . . . 72
3.4.1 Conditional specifications . . . 72
3.4.2 Conditional rewriting . . . 74

4 Orthogonality **78**
Jan Willem Klop, Vincent van Oostrom, Roel de Vrijer
4.1 Introduction . . . 78
4.2 Descendants and residuals . . . 82
4.3 Confluence via the parallel moves lemma . . . 89
4.4 Underlinable reductions . . . 91
4.5 Complete developments . . . 95
4.6 Completion of reduction diagrams . . . 99
4.6.1 Elementary diagrams . . . 100
4.6.2 Reduction diagrams . . . 101
4.6.3 Permutation equivalence . . . 106
4.7 Non-erasing reductions . . . 108
4.8 Reduction strategies . . . 112
4.8.1 Relaxing the constraints in $\mathbb{F}_{\mathrm{lm}}$, $\mathbb{F}_{\mathrm{GK}}$ and $\mathbb{F}_{\mathrm{po}}$ . . . 115

5 Properties of rewriting: decidability and modularity **119**
Jan Willem Klop, Roel de Vrijer
5.1 Decidability . . . 119
5.2 Easy (un)decidability results . . . 120
5.3 Undecidability of strong normalization . . . 121
5.3.1 Undecidability of SN via Turing machines . . . 122
5.3.2 Reducing Post's correspondence problem to SN . . . 125
5.4 Undecidability properties of combinatory logic . . . 127
5.5 Modularity . . . 128
5.5.1 Basic notions concerning disjoint sums of TRSs . . . 129
5.5.2 Modularity of weak normalization . . . 132
5.6 Modularity of confluence . . . 133
5.7 The case of strong normalization . . . 134

6 Termination **139**
Hans Zantema
6.1 Introduction . . . 139
6.2 Semantical methods . . . 141
6.2.1 Well-founded monotone algebras . . . 141
6.2.2 Polynomial interpretations . . . 144

6.2.3 Lexicographic combinations 148
6.2.4 Other examples 150

7 Completion of equational specifications **152**
Inge Bethke
7.1 Equational specifications: syntax and semantics 152
7.2 Initial algebra semantics 158
7.3 Critical pair completion 160

A Mathematical background **167**
Marc Bezem
A.1 Relations, functions and orders 167
A.2 Multisets 176

References **181**

List of notations **186**

Index of authors **190**

Index of subjects **192**

Preface

Terese Lite is an unofficial excerpt of the forthcoming monograph Term Rewriting Systems by Terese, a group of authors. (The names of the authors are listed below, in the complete overview of the chapters.) This excerpt is solely intended for use in the course Termherschrijfsystemen given at the VU, spring 2003. It contains the introduction, abridged versions of the first five chapters, a small part of Chapter 6 and about half of Chapter 7. Finally, the relevant parts of the appendix om mathematical background are included. To give a fair impression of the field of term rewriting and of the scope of the book outlines of all chapters are given below in this preface.

Nota bene: The numbering of sections, definitions, theorems, etcetera in this excerpt differs in many places from the original numbering of the complete version of Terese [2003].

The subject of term rewriting

Rewriting is the theory of stepwise, or discrete, transformations of objects, as opposed to continuous transformations of objects. Since computations in computer science are typically stepwise transformations of objects, rewriting is therefore a foundational theory of computing.

The origins of rewriting go further back than computer science, namely to mathematics and mathematical logic. Historically, the major source for the development of term rewriting is the emergence in the 1930s of the λ-calculus and its twin combinatory logic (CL), both capturing the notion of computability. The main concepts for term rewriting, confluence and termination, were analysed in the framework of these fundamental rewriting systems. An even earlier appearance of term rewriting can be found in Herbrand–Gödel computability.

Later on, the notion of a first-order term rewriting system (TRS) in its present form was formulated, and used for implementation and analysis of abstract data type specifications, regarding consistency properties, computability theory, decidability of word problems, and theorem proving. An important direction in this area started with the emergence of the completion method in the landmark paper Knuth and Bendix [1970].

The paradigm term rewriting systems of λ-calculus and CL have not only deeply influenced the theoretical side of computer science but also had a great impact on the practical side regarding the development of programming

languages: λ-calculus leading to LISP (McCarthy [1960]) and CL leading to Miranda (Turner [1986]). One can say that term rewriting provides the foundations of functional programming.

There is even more to the classical rewriting systems λ-calculus and CL: the aspect of types, leading to various typed λ-calculi that are the basis for present day proof checkers such as Coq. This book will not be concerned with the vast world of typed λ-calculi; nevertheless, several of the techniques and notions present here apply directly to the family of typed λ-calculi, see Barendregt [1992]. In contrast to the first-order rewriting system of CL, the λ-calculus is higher-order in the sense that it has bound variables. Recently, there has been a fruitful development of the general notion of higher-order rewriting; we have devoted a chapter to it.

We can state that by now the discipline of term rewriting has come of age. This is demonstrated by the yearly RTA conferences, the related TLCA conferences, the Termination workshops, the newly established IFIP W.G.1.6 on term rewriting, the European ESPRIT projects on term graph rewriting, and the appearance of the first textbooks on the subject (Avenhaus [1995], Baader and Nipkow [1998]). There is also the well-maintained RTA list of open problems, `http://www.lri.fr/~rtaloop/`, based on Dershowitz et al. [1993]. The web page of the book will provide links to some of the above-mentioned conferences and institutions.

A note on terminology. As said before, rewriting has as main origins both mathematical logic and computer science. From mathematical logic some terminology is inherited that favours the words 'reduction', 'Church–Rosser property', 'strong normalization'. Alternative terminology is inherited from computer science, where these items are called respectively 'rewriting', 'confluence property', 'termination'. In the sequel we will respect both these original influences.

The chapters

We will now present and discuss briefly the contents of the chapters. (In their original form, also of the ones not included in the excerpt.)

0. Introduction

In a short introductory chapter we sketch some basic examples of rewriting. One example, stemming from the theory of knots, does not involve terms at all. This illustrates the fact that some of the more abstract notions studied in this book have a broader scope than the manipulation of syntactic objects.

1. Abstract reduction systems

Marc Bezem, Jan Willem Klop

Many aspects of rewriting can be studied independently of the nature of the objects that are rewritten. This abstract approach is pursued in the

first chapter. We formally introduce the basic notions of abstract reduction systems, such as reduction, conversion, confluence and normalization. We study some variations and interrelations and collect several extensions in the form of exercises at the end of the chapter.

2. First-order term rewriting systems
Jan Willem Klop, Roel de Vrijer

Proper term rewriting starts here. The basic notions are formulated with several exercises. Equational reasoning is introduced. The chapter concludes with the notion of a critical pair and the Critical Pair Lemma.

3. Examples of TRSs and special rewriting formats
Jan Willem Klop, Roel de Vrijer

Here we present some extended examples, in particular that of combinatory logic. It is important since it is a very rich TRS in itself, but also because much of the theory of term rewriting goes back to the history of combinatory logic – and the related λ-calculus. We discuss Herbrand–Gödel computability as an early application of rewriting techniques. Several extensions and variations of the format of term rewriting are introduced, such as string rewriting and conditional rewriting. An interesting example of a two-sorted TRS is the $\lambda\sigma$-calculus, which, like combinatory logic, tries to capture the mechanisms of function abstraction and substitution within a first-order format.

4. Orthogonality
Jan Willem Klop, Vincent van Oostrom, Roel de Vrijer

After a formal introduction of orthogonal TRSs, the basic theorems follow: confluence, the theorems of Church and O'Donnell. The confluence theorem is proved in various ways, also using inductive proofs. The notion of a reduction strategy is defined; several strategies and their properties are discussed. The chapter concludes by establishing confluence for weakly orthogonal TRSs via the tiling method and a section on orthogonal conditional term rewriting systems.

5. Properties of rewriting: decidability and modularity
Jan Willem Klop, Roel de Vrijer

In the first half of this chapter we collect various issues concerning decidability of properties such as termination. One section is devoted to decidability questions concerning combinatory logic. In the second half of the chapter the subject of modularity is treated, dealing with the question what properties are preserved in disjoint unions of rewriting systems.

6. Termination
Hans Zantema

This chapter gives a full coverage of termination techniques, with special care taken for the well-definedness of the classical method of recursive path orders.

Also the termination hierarchy is presented. At the end, sections are devoted to the method of dependency pairs and contextual rewriting.

7. Completion of equational specifications
Inge Bethke

This chapter presents a standard introduction to critical pair completion, demonstrated with the famous example of completing the axioms for groups, with an exposition of the treatment of completion via proof-orderings, and sections on proof by consistency and E-unification.

8. Equivalence of reductions
Vincent van Oostrom, Roel de Vrijer

An advanced treatment of orthogonality is presented. Several equivalences of reductions, such as permutation equivalence and projection equivalence will be investigated in depth. The notion of a proof term plays a crucial role, as do techniques such as tracing, labelling, projection and standardization. The chapter concludes with a short treatment of braids, as an example where the methods of this chapter extend beyond the realm of term rewriting per se.

9. Strategies
Vincent van Oostrom, Roel de Vrijer

A reduction strategy determines or gives an indication of how a term should be reduced. There may be several possibilities and some may be better than others, given the objective of the strategy (mostly this is finding a normal form). Needed and fair strategies are defined and shown to be normalizing for orthogonal term rewriting systems. Thereby proofs are provided for some of the results on strategies announced in Chapter 4. Also notions of optimality and pessimality (worst case) are introduced. It is shown that needed family reduction is optimal and that the limit and zoom-in strategies are pessimal.

10. Lambda calculus
Inge Bethke

In this chapter, which can also be used as an independent concise survey, we treat the basic facts of λ-calculus, the primary example of a higher-order rewriting system. Some fundamental theorems are covered. Part of their interest in this setting derives from the fact that they have their counterparts in the theory of orthogonal rewriting. Also definability and extensions of the λ-calculus are treated, and, in the concluding section, the typed λ-calculus.

11. Higher-order rewriting
Femke van Raamsdonk

Higher-order rewriting is to be understood here as rewriting with bound variables. We discuss two frameworks of higher-order rewriting, the higher-order rewriting systems and the combinatory reduction systems, and briefly comment on other forms of higher-order rewriting. Then we survey the rewriting

theory of higher-order rewriting, focusing on the subjects of confluence and termination. We also discuss combinations of λ-calculus and TRSs.

12. Infinitary rewriting
Richard Kennaway, Fer-Jan de Vries

Here we generalize the finitary notions of term rewriting to an infinite setting. Terms may be infinitely deep, and reduction sequences may be infinitely long, with an ordinal as length. Infinitary confluence in the general orthogonal case is seen to fail – but with suitable restrictions is restored. We conclude with a section on the infinitary version of λ-calculus.

13. Term graph rewriting
Erik Barendsen

We present a short introduction to the basic notions of term graph rewriting, both in the usual style and in the equational style. A main theorem is the confluence for orthogonal term graph rewriting. At the end of the chapter the folklore proof system for μ-terms is proved to be complete for the semantics of tree unwinding, also given by bisimulation equivalence.

14. Advanced ARS theory
Marc Bezem, Jan Willem Klop, Vincent van Oostrom

This chapter presents some advanced topics in abstract rewriting: the method of decreasing diagrams to prove confluence, as well as rewriting modulo an equivalence relation (on the level of abstract rewriting).

15. Rewriting-based languages and systems
Jan Heering, Paul Klint

This chapter is devoted to applications and implementations of term rewriting. Many links to rewrite tools and projects are given. Also, there are many pointers to the various notions and techniques covered in the preceding chapters that figure in these applications.

A. Mathematical background
Marc Bezem

This appendix is intended to be used as a reference. It provides a self-contained introduction to all basic mathematical concepts and results used throughout the book.

0

Introduction

We start the book by presenting some basic examples of rewriting, in order to set the stage.

At school, many of us have been drilled to simplify arithmetical expressions, for instance:

$$(3+5)\cdot(1+2) \to 8\cdot(1+2) \to 8\cdot 3 \to 24$$

This simplification process has several remarkable properties.

First, one can perceive a direction, at least in the drill exercises, from complicated expressions to simpler ones. For this reason we use $\to$ rather than $=$, even though the expressions are equal in the sense that they all denote the same number (24). The relation $\to$ is called a *reduction relation*.

Second, in most drill exercises the simplification process yields a result in the form of an expression that cannot be simplified any further, which we call a *normal form*. In the above example the result 24 is such a normal form.

Third, the simplification process is non-deterministic, often different simplifications are possible. It is clearly desirable that all simplifications lead to the same result. Indeed the above outcome 24 can be obtained in different ways, e.g. also by

$$(3+5)\cdot(1+2) \to (3+5)\cdot 3 \to 3\cdot 3+5\cdot 3 \to 9+5\cdot 3 \to 9+15 \to 24$$

The property that simplifications can have at most one final result is called *uniqueness of normal form*.

The process of simplifying arithmetical expressions built up from numbers with operations like $+$ and $\cdot$ can be analysed (and taught) as performing *elementary steps* in *contexts*. The elementary steps are based on the tables of addition and multiplication. The contexts are arithmetical expressions with a hole, denoted by $\Box$, indicating the place where the elementary step is to take place. In the first example above, the elementary step $3+5 \to 8$ is performed in the context $\Box\cdot(1+2)$. This means that $\Box$ is $3+5$ before, and 8 after, the elementary step, yielding the simplification $(3+5)\cdot(1+2) \to 8\cdot(1+2)$. (Brackets are not part of the term, but are auxiliary symbols preventing false readings of the term.) The next elementary step is $1+2 \to 3$, performed in the context $8\cdot\Box$, yieding the simplification $8\cdot(1+2) \to 8\cdot 3$. Finally, the elementary step $8\cdot 3 \to 24$ is performed in the context $\Box$, as this elementary step involves the whole term.

Among our early experiences are, besides arithmetical simplification, also drill exercises that do not yield a final result, such as counting:

$$1 \to 2 \to 3 \to \cdots$$

Another example is the cyclic elapsing of the hours on a clock:

$$1 \to 2 \to \cdots \to 11 \to 12 \to 1 \to \cdots$$

The absence of a normal form is called *non-termination*. In many applications termination is a desirable property. The following example shows that termination can be a non-trivial question.

Define a reduction relation on positive integer numbers

$$n \to n'$$

by putting $n' = n/2$ if $n > 1$ is even, and $n' = 3n + 1$ if $n > 1$ is odd. Thus 1 is the only normal form. We have reductions such as

$$\begin{aligned} &7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \\ &\to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1 \end{aligned}$$

Note that we have both decrease (in the case $n/2$) and increase (in the case $3n + 1$). It is an open question[1] whether or not every positive integer can be reduced to 1 in this way.

This book is about the theory of stepwise, or discrete, transformations of objects, as opposed to continuous transformations of objects. Many computations, constructions, processes, translations, mappings and so on, can be modelled as stepwise transformations of objects. Clearly, this yields a large spectrum of applications, depending on what are the objects of interest and what transformations, or *rewriting*, one wishes to do. One has string rewriting, term rewriting, graph rewriting, to name some of the principal subjects that will be treated. In turn, these main subjects lead to several specialized theories, such as conditional term rewriting, infinitary term rewriting, term graph rewriting, and many more. In all these different branches of rewriting the basic concepts are the same, and known as *termination* (guaranteeing the existence of normal forms) and *confluence* (securing the uniqueness of normal forms), and many variations of them.

In order to appreciate the variety of applications and to further introduce the main concepts, let us view a few examples from different fields. We will return to some of these examples in future chapters.

[1] Collatz's problem, also known as the Syracuse problem.

0.1. An example from functional programming

Rewriting techniques can be used to specify operations on abstract data types. The first introductory example, taken from Arts [1997], describes in a concise way an algorithm for dividing natural numbers, giving for all pairs (m, n) with $n \geq 1$ and $m = n \cdot q \geq 0$ the correct result q. The abstract data type in question is that of the natural numbers built up with $\mathbf{0} : \mathbb{N}$ and $\mathbf{S} : \mathbb{N} \to \mathbb{N}$. We write 0 for $\mathbf{0}$, 1 for $\mathbf{S}(\mathbf{0})$, 2 for $\mathbf{S}(\mathbf{S}(\mathbf{0}))$, and so on. There are four rewrite rules:

$$\begin{aligned} \mathit{minus}(x, \mathbf{0}) &\to x \\ \mathit{minus}(\mathbf{S}(x), \mathbf{S}(y)) &\to \mathit{minus}(x, y) \\ \mathit{quot}(\mathbf{0}, \mathbf{S}(y)) &\to 0 \\ \mathit{quot}(\mathbf{S}(x), \mathbf{S}(y)) &\to \mathbf{S}(\mathit{quot}(\mathit{minus}(x, y), \mathbf{S}(y))) \end{aligned}$$

As an example we exhibit the following reduction sequence.

$$\begin{aligned} \mathit{quot}(4, 2) &\to \mathbf{S}(\mathit{quot}(\mathit{minus}(3, 1), 2)) \\ &\to \mathbf{S}(\mathit{quot}(\mathit{minus}(2, 0), 2)) \\ &\to \mathbf{S}(\mathit{quot}(2, 2)) \\ &\to \mathbf{S}(\mathbf{S}(\mathit{quot}(\mathit{minus}(1, 1), 2))) \\ &\to \mathbf{S}(\mathbf{S}(\mathit{quot}(\mathit{minus}(0, 0), 2))) \\ &\to \mathbf{S}(\mathbf{S}(\mathit{quot}(0, 2))) \\ &\to \mathbf{S}(\mathbf{S}(\mathbf{0})) = 2 \end{aligned}$$

Normal forms are $0, 1, 2, \ldots$, but also $\mathit{minus}(0, 1)$ and $\mathit{quot}(2, 0)$. For the correctness of such algorithms with respect to the operations on the abstract data type, it is obviously desirable that the algorithms have unique results. This is guaranteed if every reduction sequence eventually terminates, and moreover the resulting normal form is independent of the particular choice of reduction sequence. Later on in this book we will develop methods to facilitate the proofs of such properties.

0.2. An example from topology: Reidemeister moves

In this example the objects are knots. For our purposes it suffices to state that a knot is a piece of (flexible) wire whose ends coincide. We assume knots to be two-dimensional, lying on a flat surface such as a kitchen table, with crossings only in one single point. In case of a crossing, it is always clear which of the two parts of the wire involved is the upper and which is the lower (in the pictures, the lower wire is interrupted just before and after the crossing).

Knots are considered to be equivalent when they can be transformed into one another in a continuous way, that is, without breaking the wire. As a

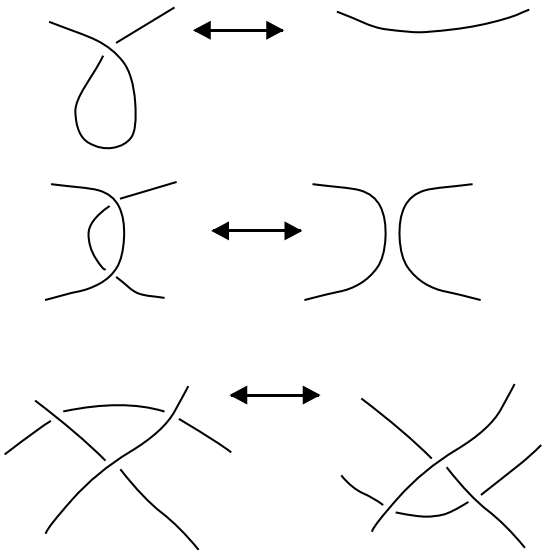


Figure 1: Reidemeister moves

consequence, knots could be taken as equivalence classes, but here we are interested in the equivalence relation itself. There are some well-known elementary transformations, known as the 'Reidemeister moves', such that two knots are equivalent if and only if they can be transformed into one another using only Reidemeister moves.

Figure 1 depicts the Reidemeister moves, Figure 2 gives a transformation between a certain knot and the trivial knot, also called the 'un-knot'. Although the un-knot is the simplest knot, it should not be considered as a normal form: the Reidemeister moves can be used in both directions. There is no use for the concept of normal form in this example. Not every knot can be transformed into the un-knot; there are in fact infinitely many different knots.

0.3. An example from logic: tautology checking

This example exploits the well-known equivalence between boolean algebras and boolean rings (rings with $x^2 = x$). In the setting of boolean algebras $\Rightarrow, \vee, \neg$ are the usual propositional connectives; $+$ is exclusive disjunction and $\cdot$ is conjunction. In the setting of rings $+$ is the ring addition, $\cdot$ the ring multiplication. Furthermore, 0 stands for the boolean false as well as for the additive unit, and 1 stands for the boolean true as well as for the multiplicative unit. The operators $+$ and $\cdot$ are supposed to be commutative and associative.

Now we have the remarkable fact that every propositional formula is a tautology if and only if it can be rewritten to 1 using the rewrite rules be-

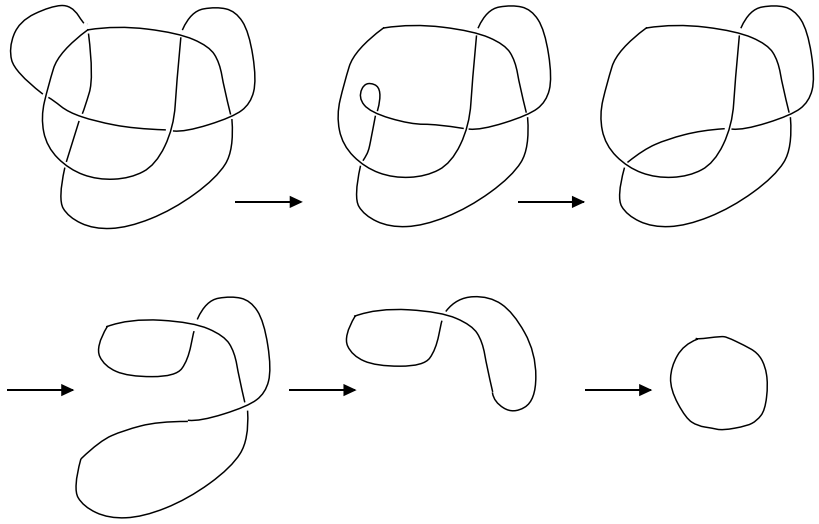


Figure 2: Un-knotting

low, *modulo associativity and commutativity*. This means that rewrite steps may be preceded by rearranging the order and the bracket structure of sums and products. (In fact this constitutes an example of rewriting modulo an equivalence relation, made precise in Chapter 14, Section 3 of Terese [2003].)

The above equivalence and the rewrite rules already appear in Herbrand's PhD thesis of 1929. The symbol $\rightarrow$ is used for a rewrite step, and should not be confused with $\Rightarrow$ which stands for implication.

$$\begin{array}{lcl}
x \Rightarrow y & \rightarrow & x \cdot y + x + 1 \\
x \vee y & \rightarrow & x \cdot y + x + y \\
\neg x & \rightarrow & x + 1 \\
x + 0 & \rightarrow & x \\
x + x & \rightarrow & 0 \\
x \cdot 0 & \rightarrow & 0 \\
x \cdot 1 & \rightarrow & x \\
x \cdot x & \rightarrow & x \\
x \cdot (y + z) & \rightarrow & x \cdot y + x \cdot z
\end{array}$$

As an example we exhibit the following reduction of the tautology $p \Rightarrow (q \Rightarrow p)$ to 1, where the most relevant associative/commutative steps are also stipulated. As usual in algebra, we omit the multiplication sign $\cdot$.

$$\begin{array}{lcl}
p \Rightarrow (q \Rightarrow p) & \rightarrow & p(q \Rightarrow p) + p + 1 \\
& \rightarrow & p(qp + q + 1) + p + 1 = p((qp + q) + 1) + p + 1 \\
& \rightarrow & p(qp + q) + p1 + p + 1 \\
& \rightarrow & pqp + pq + p1 + p + 1
\end{array}$$

$$
\begin{aligned}
&\to\ pqp + pq + p + p + 1 = pqp + pq + (p + p) + 1\\
&\to\ pqp + pq + 0 + 1 = pqp + (pq + 0) + 1\\
&\to\ pqp + pq + 1 = (pp)q + pq + 1\\
&\to\ pq + pq + 1 = (pq + pq) + 1\\
&\to\ 0 + 1 = 1 + 0\\
&\to\ 1
\end{aligned}
$$

It is not our goal to give a correctness proof of this algorithm for tautology checking. Note, once more, that rewriting is non-deterministic: we could have started in the example above with the rightmost implication in $p \Rightarrow (q \Rightarrow p)$. It is necessary for the correctness of the algorithm that any reduction sequence starting with $p \Rightarrow (q \Rightarrow p)$ yields the outcome 1. This will be guaranteed by a notion called *confluence*, formulated for arbitrary abstract reduction systems in the chapter that follows now.

1

Abstract reduction systems

Marc Bezem
Jan Willem Klop

Many aspects of rewriting can be studied independently of the nature of the objects that are rewritten. This abstract approach is expressed by the title of this chapter. We formally introduce the basic notions of abstract reduction systems such as reduction, conversion, confluence and normalization. We study some variations and interrelations and collect several extensions in the form of exercises at the end of the chapter.

In this chapter, we capture the common substratum of rewriting theory. The structures involved will be called *abstract reduction systems* or *abstract rewriting systems*. An abstract reduction system is just a set together with one or more binary relations. The set specifies the objects and the binary relations represent the transformations pertaining to the objects in an abstract way, not taking the inner structure (contexts, elementary steps) into account.

1.1. Basic notions of ARSs

In this section we shall formally introduce the basic notions of abstract rewriting. In the next section we collect several implications between the various properties of ARSs.

1.1.1. Definition (basics). An *abstract reduction system* (ARS) is a structure $\mathcal{A} = (A, \{\to_\alpha |\ \alpha \in I\})$ consisting of a set A and a set of binary relations $\to_\alpha$ on A, indexed by a set I. We write $(A, \to_1, \to_2)$ instead of $(A, \{\to_\alpha |\ \alpha \in \{1,2\}\})$. For $\alpha \in I$, the relations $\to_\alpha$ are called *reduction* or *rewrite* relations. Sometimes we will refer to $\to_\alpha$ as α. In the case of just one reduction relation, we simply write $\to$. We write $\to_I$ for the union $\bigcup\{\to_\alpha |\ \alpha \in I\}$. Reversed arrows denote the inverse relations.

In the sequel, we will consider several equivalence relations on the set A. One of them is identity on A. As the elements of A will often have a syntactical nature, we will use $\equiv$ to denote identity on A, conforming to the convention that $\equiv$ expresses syntactical identity. The usual symbol $=$ to denote identity now becomes available to denote another important equivalence relation, namely *convertibility*, the equivalence relation generated by $\to$.

1.1.2. DEFINITION (reduction). Let $\mathcal{A} = (A, \{\to_\alpha |\ \alpha \in I\})$ be an ARS and let $\alpha \in I$.

(i) If we have $(a, b) \in \to_\alpha$ for $a, b \in A$, then we write $a \to_\alpha b$ and call b a *one-step* (α-)*reduct* of a, or a a *one-step* (α-)*expansion* of b.

(ii) A *reduction sequence* with respect to $\to_\alpha$ is a finite or infinite sequence $a_0 \to_\alpha a_1 \to_\alpha a_2 \to_\alpha \cdots$. Reduction sequences are also called *reduction paths*, or *reductions* for short. Every reduction sequence has a first element, and a finite reduction sequence also has a last element. Whenever we want to stipulate these elements we use the following terminology: a reduction sequence starting from a is called a reduction sequence *of* a, and if such a reduction sequence ends in b, then it is called a reduction sequence *from* a *to* b. The element b is called an (α-)*reduct* of a in this case, which we also phrase by: a *reduces* to b.

(iii) A *reduction step* is a specific occurrence of $\to_\alpha$ in a reduction sequence.

(iv) The *length* of a finite reduction sequence is the number of reduction steps occurring in this reduction sequence (which is one less than the number of elements!).

(v) An *indexed reduction sequence* is a finite or infinite sequence of the form $a \to_\alpha b \to_\beta c \to_\gamma \cdots$ with $a, b, c, \ldots \in A$ and $\alpha, \beta, \gamma, \ldots \in I$. We write $a \to_I b$ if we do not wish to specify the index of the reduction step.

1.1.3. NOTATION. The reflexive closure of $\to_\alpha$ is $\to_\alpha^{\equiv}$. The symmetric closure of $\to_\alpha$ is $\leftrightarrow_\alpha$. The transitive closure of $\to_\alpha$ is $\to_\alpha^+$. The inverse relation of $\to_\alpha$ is $\to_\alpha^{-1}$, also denoted by $\leftarrow_\alpha$. Let $\to_\beta$ also be a reduction relation on A. The union $\to_\alpha \cup \to_\beta$ is denoted by $\to_{\alpha\beta}$. The composition $\to_\alpha$ and $\to_\beta$ is denoted by $\to_\alpha \cdot \to_\beta$. We have $a \to_\alpha \cdot \to_\beta c$ if and only if $a \to_\alpha b \to_\beta c$ for some $b \in A$.

The reflexive–transitive closure of $\to_\alpha$ is written as $\twoheadrightarrow_\alpha$.[1] We deliberately deviate from the notation $\to_\alpha^*$ introduced in Definition A.1.10, since the double arrow notation is more convenient in diagrams. We have $a \twoheadrightarrow_\alpha b$ if and only if there is a finite reduction sequence $a \equiv a_0 \to_\alpha a_1 \to_\alpha \cdots \to_\alpha a_n \equiv b$ ($n \geq 0$). If we write $\sigma : a \twoheadrightarrow_\alpha b$, then σ denotes an arbitrary reduction sequence from a to b, also called a *reduction* from a to b. We write $\sigma : a \to_\alpha a_1 \to_\alpha \cdots \to_\alpha b$ whenever we want to stipulate a specific reduction sequence σ from a to b. Similarly, finite indexed reduction sequences will be denoted by $\sigma : a \twoheadrightarrow_I b$.

1.1.4. DEFINITION (conversion). Let $\mathcal{A} = (A, \{\to_\alpha |\ \alpha \in I\})$ be an ARS and let $\alpha \in I$.

(i) The *convertibility relation* $=_\alpha$ is defined as the equivalence relation generated by $\to_\alpha$.

[1] There is a threat of ambiguity here. With $\twoheadrightarrow_{\alpha\beta}$ we mean the reflexive–transitive closure of $\to_{\alpha\beta}$, and *not* the union $\twoheadrightarrow_\alpha \cup \twoheadrightarrow_\beta$.

(ii) A *conversion sequence* with respect to $\to_\alpha$ is a reduction sequence with respect to $\leftrightarrow_\alpha$, that is, a sequence $a_0 \leftrightarrow_\alpha a_1 \leftrightarrow_\alpha a_2 \leftrightarrow_\alpha \cdots$, with $\leftrightarrow_\alpha = \to_\alpha \cup \leftarrow_\alpha$, the symmetric closure of $\to_\alpha$.

1.1.5. NOTATION. If $a =_\alpha b$, then we say that a and b are *convertible* with respect to $\to_\alpha$. We have $a =_\alpha b$ if and only if there is a finite conversion sequence $a \equiv a_0 \leftrightarrow_\alpha a_1 \leftrightarrow_\alpha \cdots \leftrightarrow_\alpha a_n \equiv b$ $(n \geq 0)$. By a *conversion* we usually mean a pair $a, b \in A$ with $a =_\alpha b$, without stipulating the intermediate $a_0, \ldots, a_n$.

1.1.6. DEFINITION (sub-ARS). Let $\mathcal{A} = (A, \to_\alpha)$ and $\mathcal{B} = (B, \to_\beta)$ be two ARSs. Then $\mathcal{A}$ is a *sub-ARS* of $\mathcal{B}$, denoted by $\mathcal{A} \subseteq \mathcal{B}$, if the following conditions are satisfied.

(i) $A \subseteq B$;
(ii) α is the restriction of β to A, that is, $\forall a, a' \in A\ (a \to_\beta a' \Leftrightarrow a \to_\alpha a')$, or $\to_\alpha = \to_\beta \cap A^2$ more shortly;
(iii) A is closed under β, i.e. $\forall a \in A\ \forall b \in B\ (a \to_\beta b \Rightarrow b \in A)$.

If $\mathcal{A}$ is a sub-ARS of $\mathcal{B}$, then $\mathcal{B}$ is also called an *extension* of $\mathcal{A}$. If $A \subseteq B$, then the sub-ARS *generated by* A is $\mathcal{G}(A, \beta) = (A_\beta, \to_\alpha)$ with $A_\beta = \{b \in B \mid \exists a \in A\ a \twoheadrightarrow_\beta b\}$ and $a \to_\alpha a'$ if and only if $a, a' \in A_\beta$ and $a \to_\beta a'$. Thus A_β consists of all β-reducts of elements of A, and $\to_\alpha$ is the reduction relation $\to_\beta$ restricted to this set of reducts.

Of particular interest is the sub-ARS determined by an element $a \in A$ in an ARS $\mathcal{A} = (A, \to)$, the reduction graph of a.

1.1.7. DEFINITION. Let $\mathcal{A} = (A, \to)$ be an ARS, and $a \in A$. The *reduction graph* of a is $\mathcal{G}(\{a\}, \to)$, the sub-ARS of $\mathcal{A}$ generated by the singleton subset $\{a\}$. We abbreviate $\mathcal{G}(\{a\}, \to)$ by $\mathcal{G}(a)$. Thus $\mathcal{G}(a)$ has as elements all reducts of a (including a itself) and has as reduction relation $\to$ restricted to this set of reducts.

Below we will define a number of properties of the reduction relation $\to$. If this reduction relation has a certain property, then we will attribute this property also to the ARS $(A, \to)$ in question, and vice versa. Most of the properties are first defined elementwise, that is, as a property of elements of the ARS.

As we have seen in the introductory examples in Sections 0.1 and 0.3, it may be crucial whether normal forms are unique. This uniqueness is implied, as we will see in Theorem 1.2.2(i), by the even more fundamental property of confluence. Confluence of an object means that every two reduction sequences starting with that object can be prolonged to a common reduct. We will now formally introduce the notion of confluence and several notions that are closely related.

1.1.8. Definition (confluence). Let $\mathcal{A} = (A, \{\to_\alpha |\ \alpha \in I\})$ be an ARS, $\alpha, \beta \in I$ and let $\to = \to_\alpha$.

(i) We say $\to_\alpha$ *commutes weakly* with $\to_\beta$ if the diagram of Figure 1.1(a) holds, i.e. if $\forall a, b, c \in A\ (c \leftarrow_\beta a \to_\alpha b \Rightarrow \exists d \in A\ c \twoheadrightarrow_\alpha d \twoheadleftarrow_\beta b)$.

(ii) We say $\to_\alpha$ *commutes*[2] with $\to_\beta$ if $\twoheadrightarrow_\alpha$ and $\twoheadrightarrow_\beta$ commute weakly.

(iii) $a \in A$ is *weakly confluent* if $\forall b, c \in A\ (c \leftarrow a \to b \Rightarrow \exists d \in A\ c \twoheadrightarrow d \twoheadleftarrow b)$. The reduction relation $\to$ is *weakly confluent* or *weakly Church–Rosser* (WCR) if every $a \in A$ is weakly confluent, see Figure 1.1(b). Alternatively, weak confluence is called *local confluence*.

(iv) $a \in A$ is *subcommutative* if $\forall b, c \in A\ (c \leftarrow a \to b \Rightarrow \exists d \in A\ c \to^{\equiv} d \leftarrow^{\equiv} b)$. The reduction relation $\to$ is *subcommutative* (notation $\mathrm{CR}^{\leq 1}$) if every $a \in A$ is subcommutative, see the diagram in Figure 1.1(c).

(v) $a \in A$ has the *diamond property* if $\forall b, c \in A\ (c \leftarrow a \to b \Rightarrow \exists d \in A\ c \to d \leftarrow b)$. The reduction relation $\to$ has the *diamond property* (notation DP) if every $a \in A$ has the *diamond property*.

(vi) $a \in A$ has the *triangle property* if $\exists a' \in A\ \forall b \in A\ (a \to b \Rightarrow b \to a')$. (The name 'triangle property' refers to cases in which we have $a \to a'$.) The reduction relation $\to$ has the *triangle property* (notation TP) if every $a \in A$ has the *triangle property*.

(vii) $a \in A$ is *confluent* if $\forall b, c \in A\ (c \twoheadleftarrow a \twoheadrightarrow b \Rightarrow \exists d \in A\ c \twoheadrightarrow d \twoheadleftarrow b)$. The reduction relation $\to$ is *confluent* or *Church–Rosser*, or has the Church–Rosser property (CR), if every $a \in A$ is confluent, see Figure 1.1(d).

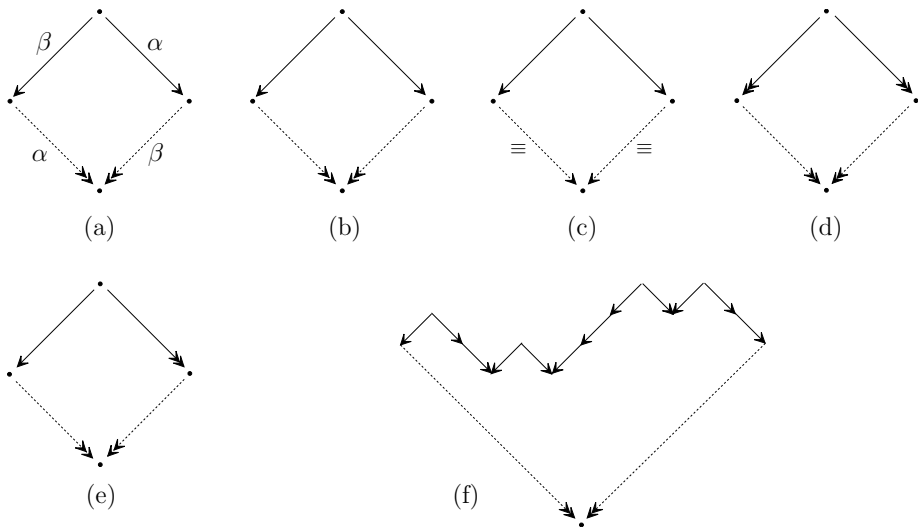


Figure 1.1: Various confluence patterns

[2]The terminology here differs from that of Bachmair and Dershowitz [1986], where α commutes with β if and only if $\beta^{-1} \cdot \alpha \subseteq \alpha \cdot \beta^{-1}$, which implies $(\beta^*)^{-1} \cdot \alpha^* \subseteq \alpha^* \cdot (\beta^*)^{-1}$, the latter being equivalent to α^* commutes with β^* in our definition.

In the following proposition we state some alternative characterizations of various notions of confluence defined above. These characterizations are algebraic in the sense that they are expressed in terms of compositions of relations. Terse as they are, they are preferably visualized by Figure 1.1. We omit the easy equivalence proofs.

1.1.9. Proposition. *Let conditions be as in Definition 1.1.8. Then*

(i) $\to_\alpha$ *commutes weakly with* $\to_\beta$ *if and only if* $\leftarrow_\beta \cdot \to_\alpha \subseteq \twoheadrightarrow_\alpha \cdot \twoheadleftarrow_\beta$,
(ii) $\to_\alpha$ *commutes with* $\to_\beta$ *if and only if* $\twoheadleftarrow_\beta \cdot \twoheadrightarrow_\alpha \subseteq \twoheadrightarrow_\alpha \cdot \twoheadleftarrow_\beta$,
(iii) $\to$ *is weakly confluent (or* WCR*) if and only if* $\leftarrow \cdot \to \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$, *that is,* $\to$ *is weakly self-commuting,*
(iv) $\to$ *is subcommutative (or* $\mathrm{CR}^{\leq 1}$*) if and only if* $\leftarrow \cdot \to \subseteq \to^{\equiv} \cdot \leftarrow^{\equiv}$,
(v) $\to$ *has the diamond property (or* DP*) if and only if* $\leftarrow \cdot \to \subseteq \to \cdot \leftarrow$ *(in case* $\to$ *is reflexive, then* $\mathrm{CR}^{\leq 1}$ *is equivalent to* DP*),*
(vi) $\to$ *is confluent (or* CR*) if and only if* $\twoheadleftarrow \cdot \twoheadrightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$.

The property WCR is often called 'local confluence'. In the sequel we will use the terms 'confluent' and 'Church–Rosser' or 'CR' without preference. Likewise for 'weakly confluent' and 'WCR'. The following proposition follows immediately from the definitions. Note especially the equivalence of (i) and (vi). Some authors call Definition 1.1.8(vii) 'confluent' and Proposition 1.1.10(vi) 'Church–Rosser'.

1.1.10. Proposition. *For every ARS* $(A, \to)$*, the following are equivalent:*

(i) $\to$ *is confluent;*
(ii) $\twoheadrightarrow$ *is weakly confluent;*
(iii) $\twoheadrightarrow$ *is self-commuting;*
(iv) $\twoheadrightarrow$ *is subcommutative (or has the diamond property, as* $\twoheadrightarrow$ *is reflexive);*
(v) $\leftarrow \cdot \twoheadrightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$, *see the diagram in Figure 1.1(e);*
(vi) $= \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$, *see the diagram in Figure 1.1(f).*

Proof. The equivalence of (i)–(iv) follows immediately from the definitions, using that $\twoheadrightarrow$ is reflexive and transitive, and hence equals $\twoheadrightarrow^{\equiv}$ and $\twoheadrightarrow^*$. Since $=$ includes $\twoheadleftarrow \cdot \twoheadrightarrow$, which in turn includes $\leftarrow \cdot \to$, (vi) implies (i), and (i) implies (v) in turn. It remains to prove that (v) implies (vi). By definition (vi) expresses $\forall a, b \in A \ (a = b \Rightarrow \exists c \in A \ a \twoheadrightarrow c \twoheadleftarrow b)$. Assume (v) and let $a = b$. Recall that $=$ is the equivalence generated by $\to$. Hence there exist $a_0, \ldots, a_n \in A$ $(n \geq 0)$ such that $a \equiv a_0 \leftrightarrow \cdots \leftrightarrow a_n \equiv b$. We argue by induction on n. If $n = 0$, then we trivially have $a \twoheadrightarrow c \twoheadleftarrow b$ by taking $c \equiv a \equiv b$. Assume (vi) has been proved for n and let $a \equiv a_0 \leftrightarrow \cdots \leftrightarrow a_n \leftrightarrow a_{n+1} \equiv b$. By the induction hypothesis there exists $c \in A$ such that $a \twoheadrightarrow c \twoheadleftarrow a_n$. If $a_{n+1} \to a_n$, then we are done. If $a_n \to a_{n+1}$, then we can apply (v) to $a_{n+1} \leftarrow a_n \twoheadrightarrow c$ and we are also done. This completes the proof of (vi). □

The diamond property and the triangle property play an important role in many proofs of CR. Note that in general we have TP $\Rightarrow$ DP. The typical applications make use of the following simple but fundamental proposition.

1.1.11. PROPOSITION. *Consider an ARS* $(A, \to)$. *Let* $\to\!\!\diamond\!\!\to$ *be a reduction relation on* A *such that:*

(i) $\to \subseteq \to\!\!\diamond\!\!\to \subseteq \twoheadrightarrow$.

(ii) $\to\!\!\diamond\!\!\to$ *satisfies* DP *(or* TP*).*

Then $\to$ *is* CR.

PROOF. See Exercise 1.3.1 □

1.1.12. REMARK. Note that if the diamond property (DP) holds for $\to$ and $a \to b$, then b cannot be a normal form. As a matter of fact, in all or almost all cases where DP plays a role it will concern a reflexive relation, for which there are no normal forms. For reflexive relations DP is equivalent with the property CR^1, to be defined in Exercise 1.3.8 below.[3]

After the above introduction of confluence, we now turn to the second fundamental notion concerning ARSs, namely that of termination or normalization. This notion comes in two equally important forms, called *weak* and *strong* normalization. Although the strong form logically implies the weak form, applications usually require only the weak form. An object is weakly normalizing if it reduces to a normal form, that is, there exists a reduction sequence starting with that object and ending in a normal form. An object is strongly normalizing if there exists no infinite reduction sequence staring with that object. The introductory example on abstract data types in Section 0.1 has the property of strong normalization. We will now formally introduce the notion of normalization and several notions that are closely related.

1.1.13. DEFINITION (normalization). Let $\mathcal{A} = (A, \to)$ be an ARS.

(i) $a \in A$ is a *normal form* if there exists no $b \in A$ such that $a \to b$.

(ii) $a \in A$ is *weakly normalizing*[4] if $a \twoheadrightarrow b$ for some normal form $b \in A$. The reduction relation $\to$ is *weakly normalizing* (WN) if every $a \in A$ is weakly normalizing.

(iii) $a \in A$ is *strongly normalizing* if every reduction sequence starting from a is finite. The reduction relation $\to$ is *strongly normalizing* (SN), or *terminating*, *noetherian*, if every $a \in A$ is strongly normalizing.

---

[3] In Dershowitz and Jouannaud [1990] the property CR^1 is called *strong confluence*. As remarked there, confluence of a rewrite relation $\to$ is the same as strong confluence (or DP, for that matter) of $\twoheadrightarrow$. We rather avoid the terminology strong confluence, since in Huet [1980] it is used in yet a different way.

[4] Sometimes one says just 'normalizing', instead of 'weakly normalizing', e.g. in Baader and Nipkow [1998].

(iv) We say $\to$ has the *normal form property* (NF) if $a = b \Rightarrow a \twoheadrightarrow b$ for all $a \in A$ and any normal form $b \in A$.

(v) We say $\to$ has the *unique normal form property* (UN) if $a = b \Rightarrow a \equiv b$ for all normal forms $a, b \in A$.

(vi) We say $\to$ has the *unique normal form property with respect to reduction* ($\mathrm{UN}^{\to}$) if $a \twoheadleftarrow \cdot \twoheadrightarrow b \Rightarrow a \equiv b$ for all normal forms $a, b \in A$.

An alternative characterization of $\to$ is SN is: $\leftarrow$ is well-founded (WF), see Definition A.1.5.

A reduction sequence is called *maximal* if it cannot be prolonged. This means that maximal reduction sequences either are infinite, or end in a normal form. Given an ARS $\mathcal{A} = (A, \to)$, every $a \in A$ has a maximal reduction sequence by the following construction (using the Axiom of Choice!). If a is a normal form itself, then the empty reduction sequence is maximal. Otherwise, there exists $b \in A$ such that $a \to b$. Repeating this argument with b instead of a yields either an infinite reduction sequence, or a reduction sequence ending in a normal form. In both cases we obtain a maximal reduction sequence of a. Now SN can equivalently be reformulated as: all maximal reduction sequences are finite. We immediately obtain WN as: there exists a finite maximal reduction sequence for every $a \in A$.

The following definition combines notions of confluence and termination into the central notion of (semi-)complete ARS.

1.1.14. Definition ((semi-)complete). Let $\mathcal{A} = (A, \to)$ be an ARS.

(i) We say $\to$ is *complete*, or *canonical*, *uniquely terminating*, if $\to$ is confluent and terminating ($\mathrm{CR} \wedge \mathrm{SN}$).

(ii) We say $\to$ is *semi-complete*, if $\to$ has the unique normal form property and is weakly normalizing ($\mathrm{UN} \wedge \mathrm{WN}$). If $\mathcal{A}$ is semi-complete, then every $a \in A$ reduces to a unique normal form denoted by $nf(a)$.

1.1.15. Definition (miscellaneous). Let $\mathcal{A} = (A, \to)$ be an ARS.

(i) We say $\to$ is *inductive* (Ind) if for every (possibly infinite) reduction sequence $a_0 \to a_1 \to \cdots$ there is an $a \in A$ such that $a_n \twoheadrightarrow a$ for all n.

(ii) We say $\to$ is *increasing* (Inc) if there is a mapping $|-| : A \to \mathbb{N}$ such that $\forall a, b \in A\ (a \to b \Rightarrow |a| < |b|)$.

(iii) We say $\to$ is *finitely branching* (FB) if for all $a \in A$ the set $\{b \in A \mid a \to b\}$ of one-step reducts of a is finite. In Huet [1980], FB is called *locally finite*.

(iv) Let $B \subseteq A$. Then B is *cofinal* in $\mathcal{A}$ if $\forall a \in A\ \exists b \in B\ a \twoheadrightarrow b$. We say that $\to$ has the *cofinality property* (CP) if in every reduction graph $\mathcal{G}(a)$, $a \in A$, there is a (finite or infinite) reduction sequence $a \equiv a_0 \to a_1 \to \cdots$ such that $\{a_n \mid n \geq 0\}$ is cofinal in $\mathcal{G}(a)$.

Note that all properties of ARSs introduced so far (CR, WCR, $\mathrm{CR}^{\leq 1}$, DP, WN, SN, WF, UN, NF, Ind, Inc, FB, CP) are preserved downwards: e.g. if $\mathcal{A} \subseteq \mathcal{B}$ and $\mathcal{B}$ is CR, then also $\mathcal{A}$ is CR (see Exercise 1.3.9).

1.2. Interrelations between ARS properties

In this section we collect several implications between the various properties of ARSs. Theorem 1.2.2(i), CR $\Rightarrow$ NF $\Rightarrow$ UN, is actually the main motivation for the concept of confluence. Confluence guarantees that normal forms are unique, which is of course a desirable state of affairs, for example in (implementations of) algebraic data type specifications. Note that Theorem 1.2.2(ii), WN $\wedge$ UN $\Rightarrow$ CR, implies that CR, NF and UN are equivalent in case WN holds. Apart from these fundamental implications, the most important fact is Theorem 1.2.1, SN $\wedge$ WCR $\Rightarrow$ CR, also known as Newman's Lemma [1942].

Of course we have the implication CR $\Rightarrow$ WCR. However, the converse does not hold. As counterexample, consider the ARS $(\{a, b, c, d\}, \rightarrow)$ depicted in Figure 1.2 (attributed by Hindley to Kleene). Observe that WCR holds, but a and d have no common reduct.

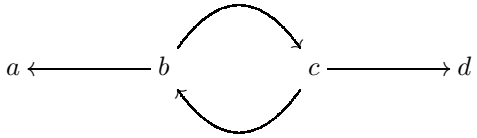


Figure 1.2: Counterexample against WCR $\Rightarrow$ CR

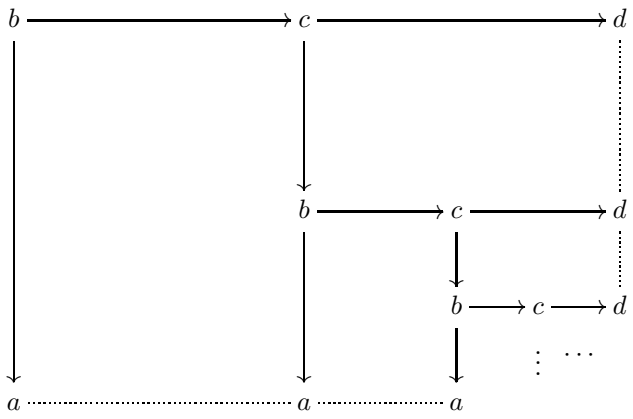


Figure 1.3: Tiling analysis of the counterexample

One might further analyse why the implication WCR $\Rightarrow$ CR actually fails, by using WCR again and again in order to find a common reduct of a and d.

This can be viewed as a tiling procedure, trying to cover completely the rectangle spanned by the reduction sequences $b \to a$ (left edge) and $b \to c \to d$ (upper edge) as depicted in Figure 1.3. (The dotted lines represent reduction sequences of length 0, to keep things orthogonal.) This view provides a valuable insight: when such a tiling procedure fails to yield a finite covering of the rectangle, an infinite reduction must necessarily arise. In Chapter 14 of Terese [2003] infinite reduction diagrams such as the one above will be analysed extensively. Thus we arrive at the first non-trivial and highly useful result.

1.2.1. THEOREM (Newman's Lemma). *Every ARS satisfies* $\mathrm{SN} \wedge \mathrm{WCR} \Rightarrow \mathrm{CR}$.

PROOF. We will give two proofs here, listed below under (i) and (ii). The first proof (Barendregt [1984], Proposition 3.1.25) is self-contained. The second proof (see Figure 1.5) using multisets will play a role in various other results in the sequel, such as Exercise 1.3.2.

Assume $\mathcal{A} = (A, \to)$ has the properties SN and WCR.

(i) By SN every $a \in A$ reduces to at least one normal form. It suffices for CR to show that this normal form is unique. Call a *ambiguous* if a reduces to two (or more) distinct normal forms. We claim that every ambiguous element has a one-step reduct which is again ambiguous. Then it follows by SN that there are no ambiguous elements, so CR must hold. In order to prove the claim, let a be ambiguous, reducing to different normal forms $n_1,\ n_2$, so $a \to b \to \cdots \to n_1$ and $a \to c \to \cdots \to n_2$ and $n_1 \not\equiv n_2$. Applying WCR to the diverging reduction steps $a \to b$, $a \to c$ yields a common reduct d such that $b \twoheadrightarrow d$ and $c \twoheadrightarrow d$. Any normal form n of d is distinct from at least one of $n_1,\ n_2$, so at least one of b, c is ambiguous. See Figure 1.4. This proves the claim.

(ii) Recall that $=$ is the equivalence generated by $\to$. Let $a = b$. Then there exist $a_0, \ldots, a_n \in A$ $(n \geq 0)$ such that $a \equiv a_0 \leftrightarrow \cdots \leftrightarrow a_n \equiv b$, where $\leftrightarrow$ is the symmetric closure of $\to$. We view $a_0 \leftrightarrow \cdots \leftrightarrow a_n$ as a *landscape* with *peaks* $a_{i-1} \leftarrow a_i \to a_{i+1}$, *valleys* $a_{i-1} \to a_i \leftarrow a_{i+1}$ and *slopes* $a_i \to \cdots \to a_{i+k}$ or $a_i \leftarrow \cdots \leftarrow a_{i+k}$, for some $k > 0$. If $a_0 \leftrightarrow \cdots \leftrightarrow a_n$ contains no peaks, then it is either one single point or one slope, or two slopes with one valley. In all these cases we immediately have $c \in A$ with $a \twoheadrightarrow c \twoheadleftarrow b$. If $a_0 \leftrightarrow \cdots \leftrightarrow a_n$ does contain a peak, say $a_{i-1} \leftarrow a_i \to a_{i+1}$, then we can eliminate this peak by applying WCR: for suitable $c_1, \ldots, d, c_1', \ldots \in A$ we have $a_{i-1} \to c_1 \to \cdots \to d \leftarrow \cdots \leftarrow c_1' \leftarrow a_{i+1}$. Then the landscape becomes $a_0 \leftrightarrow \cdots \leftrightarrow a_{i-1} \leftrightarrow c_1 \leftrightarrow \cdots \leftrightarrow d \leftrightarrow \cdots \leftrightarrow c_1' \leftrightarrow a_{i+1} \leftrightarrow \cdots \leftrightarrow a_n$. See Figure 1.5 for an example. This doesn't seem to help very much. However, we shall argue that this procedure of eliminating peaks must terminate, so that we necessarily end up with a landscape between a_0 and a_n containing no peaks. Then we will have proved CR as above. The argument uses multisets

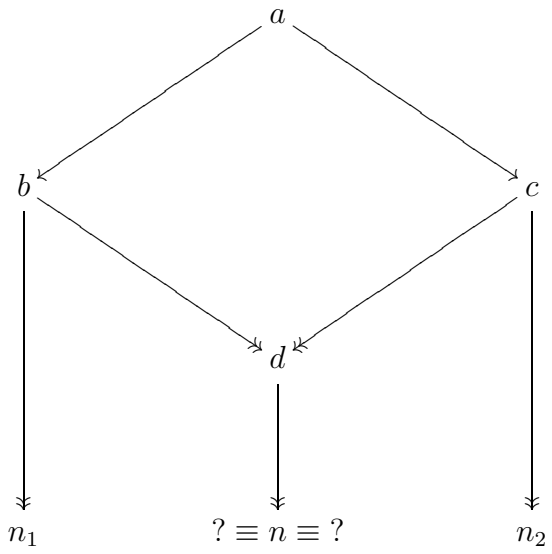


Figure 1.4: Ambiguous a has ambiguous reduct b or c

as defined in Section A.2. To a landscape $d_0 \leftrightarrow \cdots \leftrightarrow d_n$ we can associate the multiset $[d_0, \ldots, d_n]$. As $\to$ is SN, $\leftarrow$ is WF, and hence $\leftarrow^+$ is a well-founded order by Exercise A.1.11. By Theorem A.2.5, the corresponding multiset order $\leftarrow^+_\#$ is also well-founded. Now we observe that in the procedure of eliminating peaks the multisets associated to landscapes descend in the sense of this multiset order. For example, $[a_0, \ldots, a_{i-1}, c_1, \ldots, d, \ldots, c'_1, a_{i+1}, \ldots, a_n]$ originates from $[a_0, \ldots, a_n]$ by replacing a_i by the multiset $[c_1, \ldots, d, \ldots, c'_1]$ of strictly smaller elements. It follows that the procedure of eliminating peaks must terminate. □

1.2.2. Theorem. *For every ARS we have the following implications.*

(i) CR $\Rightarrow$ NF $\Rightarrow$ UN.

(ii) WN $\wedge$ UN $\Rightarrow$ CR, *i.e., every semi-complete ARS is confluent.*

(iii) $\mathrm{CR}^{\leq 1} \Rightarrow$ CR.

Proof. Let $\mathcal{A} = (A, \to)$ be an ARS.

(i) Immediate from the definitions and Proposition 1.1.10(vi).

(ii) Assume $\mathcal{A}$ satisfies WN $\wedge$ UN, so $\mathcal{A}$ is semi-complete and every $a \in A$ reduces to a unique normal form $nf(a)$. We prove confluence in the formulation of Proposition 1.1.10(vi). If $a = b$, then $nf(a) = a = b = nf(b)$, so $nf(a) = nf(b)$ and hence $nf(a) \equiv nf(b)$ by UN, which gives the desired common reduct of a and b.

(iii) Assume $\mathrm{CR}^{\leq 1}$. One easily proves CR in the formulation of Proposition 1.1.10(v) by induction on the length of the reduction sequence for $a \twoheadrightarrow b$. □

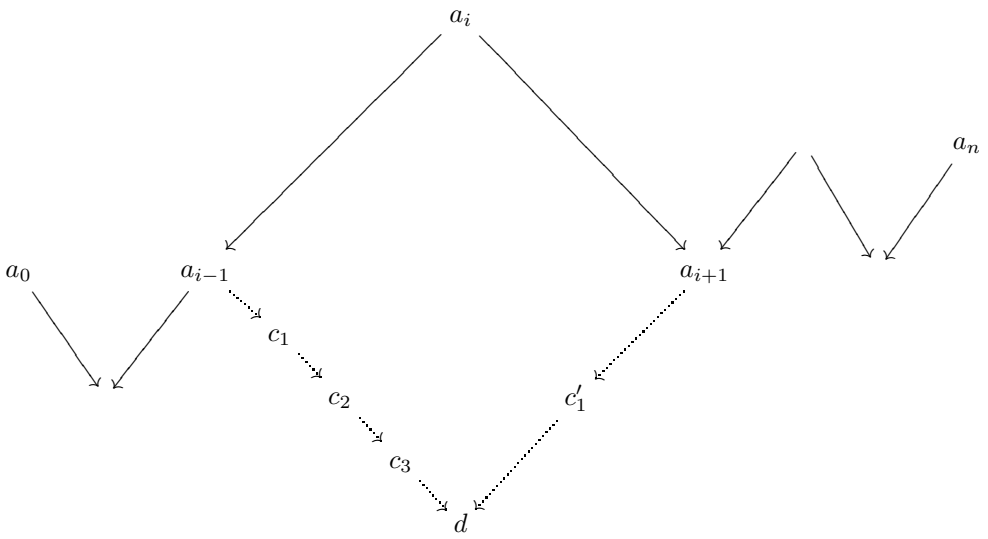


Figure 1.5: Replacing a peak by a valley

1.2.3. THEOREM. *For every ARS we have the following implications.*

(i) WN $\wedge$ UN $\Rightarrow$ Ind*, i.e., every semi-complete ARS is inductive.*

(ii) Ind $\wedge$ Inc $\Rightarrow$ SN.

(iii) WCR $\wedge$ WN $\wedge$ Inc $\Rightarrow$ SN.

PROOF. Let $\mathcal{A} = (A, \to)$ be an ARS.

(i) Immediate from the definitions.

(ii) Obvious, see Nederpelt [1973].

(iii) Assume $\mathcal{A}$ has the properties WCR and WN. Let a_0 be not SN and let $\sigma : a_0 \twoheadrightarrow n$ be a reduction sequence from a_0 to a normal form n of a_0. In particular n is SN, and we can go back along σ from n to a_0 till we arrive at an element which is *not* SN, say a_1. It follows that there are an infinite reduction sequence $a_1 \to a_2 \to \cdots$ and an element b which is SN such that $a_1 \to b \twoheadrightarrow n$. Applying WCR to a_1 yields a common reduct c of b and a_2 (see Figure 1.6). As $\mathcal{G}(b)$ is SN and WCR, it follows by Newman's Lemma 1.2.1 that b is CR, so n and c have a common reduct, which must be n itself (being a normal form). We have that a_2 is not SN, $a_2 \twoheadrightarrow c \twoheadrightarrow n$ and n is a normal form of a_2. Hence we can repeat the argument with a_2 instead of a_0, and so on. As $a_0 \twoheadrightarrow a_1 \to a_2$ contains at least one reduction step, we construct in this way an infinite reduction sequence of elements all reducing to n. This obviously conflicts with Inc, but even with a weaker version of Inc where $|-|$ is only weakly increasing, but unbounded on infinite reduction sequences. By contraposition, the assumptions WN, WCR and (weak) Inc together imply SN.

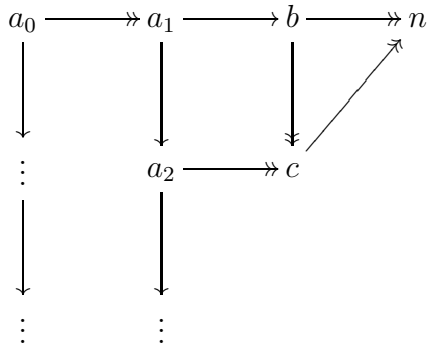


Figure 1.6: WCR $\wedge$ WN $\wedge$ Inc $\Rightarrow$ SN

The implications in the above theorems are displayed in Figure 1.7; here it is important whether an implication arrow points to the conjunction sign, or to one of the conjuncts. In the latter case *only* the conjunct in question is implied, whereas in the former case the *whole* conjunction is implied. Similarly, the exact point of departure of the tail of an implication arrow should be taken into account. For example, WN$\wedge$UN $\Rightarrow$ Ind, SN $\wedge$ WCR $\Rightarrow$ WN $\wedge$ UN, Inc $\Rightarrow$ WF, CR $\Rightarrow$ UN, but not CR $\Rightarrow$ WN $\wedge$ UN.

It is not possible to reverse any of the arrows in Figure 1.7. There are several other useful facts about ARSs. We present them in the form of exercises in Section 1.3.

1.3. Specific ARS properties

1.3.1. Confluence

1.3.1. Exercise. Consider an ARS $(A, \to)$. Let $\to\!\!\diamond\!\!\leftarrow$ be a reduction relation on A such that $\to \subseteq \to\!\!\diamond\!\!\leftarrow \subseteq \twoheadrightarrow$ and satisfying DP (or TP). Prove the following.

(i) $\to\!\!\diamond\!\!\leftarrow$ is CR.

(ii) $\twoheadrightarrow = \to\!\!\diamond\!\!\leftarrow\!\!\!\!\twoheadrightarrow$.

(iii) $\to$ is CR.

1.3.2. Exercise. Let $(A, \to_1, \to_2)$ be an ARS such that $\to_1$ and $\to_2$ commute weakly and $\to_{12}$ is SN; then $\to_1$ and $\to_2$ commute. Show also that the condition $\to_{12}$ is SN cannot be weakened to $\to_1$ is SN and $\to_2$ is SN.

1.3.3. Exercise (Rosen [1973]). If $(A, \to_1, \to_2)$ is an ARS such that $\twoheadrightarrow_1 = \twoheadrightarrow_2$ and $\to_1$ is subcommutative, then $\to_2$ is confluent.

1.3.4. Exercise (Hindley [1964]). Let $(A, \{\to_\alpha |\ \alpha \in I\})$ be an ARS such that for all $\alpha, \beta \in I$, $\to_\alpha$ commutes with $\to_\beta$. (In particular, $\to_\alpha$ commutes with itself.) Then the union $\to\ = \bigcup\{\to_\alpha |\ \alpha \in I\}$ is confluent. (This proposition is usually

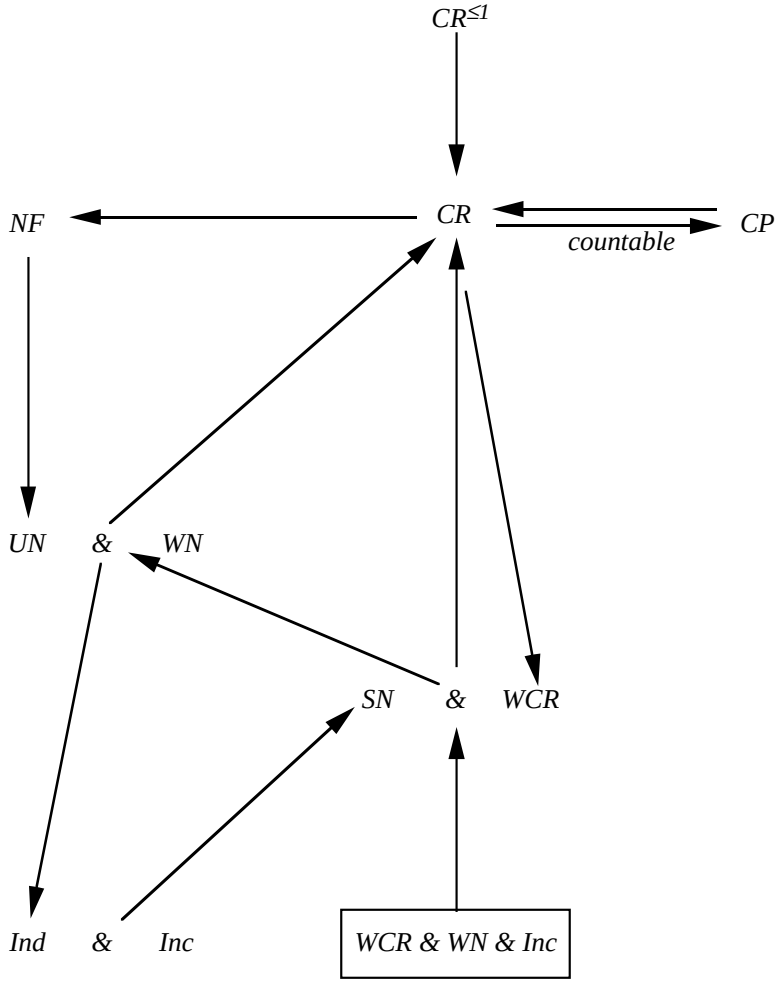


Figure 1.7: Summary of interrelated ARS properties

referred to as the Lemma of Hindley and Rosen; see e.g. Barendregt [1984], Proposition 3.3.5.)

1.3.5. Exercise (Hindley [1964]). Let $(A, \to_1, \to_2)$ be an ARS and suppose $\forall a, b, c \in A\ (b \leftarrow_1 a \to_2 c \Rightarrow \exists d \in A\ b \twoheadrightarrow_2 d \leftarrow\!\!\!\!\!=_1 c)$. Then $\to_1$ and $\to_2$ commute.

1.3.6. Exercise (Staples [1975]). Let $(A, \to_1, \to_2)$ be an ARS and suppose $\forall a, b, c \in A\ (b \leftarrow_1 a \twoheadrightarrow_2 c \Rightarrow \exists d \in A\ b \twoheadrightarrow_2 d \twoheadleftarrow_1 c)$. Then $\to_1$ and $\to_2$ commute.

1.3.2. Normalization

1.3.7. Exercise. Let $\mathcal{A} = (A, \to)$ be an ARS. If $\mathcal{A}$ satisfies SN and FB, then every reduction graph in $\mathcal{A}$ is finite.

1.3.8. Exercise (Newman [1942]). Let CR^1 be the following property of ARSs $(A, \to)$: $\forall a, b, c \in A\ (c \leftarrow a \to b \wedge b \not\equiv c \Rightarrow \exists d \in A\ c \to d \leftarrow b)$. Prove that $\mathrm{CR}^1 \wedge \mathrm{WN} \Rightarrow \mathrm{SN}$, and give a counterexample to the implication $\mathrm{CR}^{\leq 1} \wedge \mathrm{WN} \Rightarrow \mathrm{SN}$.

1.3.3. Miscellaneous

1.3.9. Exercise. Let $\mathcal{A} = (A, \to_\alpha)$ be an ARS. Define: $\mathcal{A}$ is *consistent* if not every pair of elements in A is convertible. Note that, if $\mathcal{A}$ is confluent and has two different normal forms, then $\mathcal{A}$ is consistent. Furthermore, let $\mathcal{B} = (B, \to_\beta)$ be an ARS such that $\mathcal{B}$ is an extension of $\mathcal{A}$. Then we define: $\mathcal{B}$ is *conservative over* $\mathcal{A}$, or a *conservative extension* of $\mathcal{A}$, if $\forall a, a' \in A\ (a =_\beta a' \Leftrightarrow a =_\alpha a')$. Assume that $\mathcal{B}$ is an extension of $\mathcal{A}$.

(i) Show that $\mathcal{B}$ is consistent whenever $\mathcal{B}$ is conservative over $\mathcal{A}$ and $\mathcal{A}$ is consistent.

(ii) Show that $\mathcal{A}$ is confluent whenever $\mathcal{B}$ is confluent.

(iii) Show that a confluent extension $\mathcal{B}$ of $\mathcal{A}$ is conservative.

1.3.10. Exercise. (i) Show $\mathrm{UN} \Rightarrow \mathrm{UN}^{\to}$ for every ARS, but not conversely.

(ii) Give an example of an ARS which is WCR, $\mathrm{UN}^{\to}$, but not UN.

(iii) Find a counterexample to $\mathrm{WCR} \wedge \mathrm{WN} \Rightarrow \mathrm{Ind}$.

2

First-order term rewriting systems

Jan Willem Klop
Roel de Vrijer

This chapter introduces first-order term rewriting systems and basic notions such as terms, occurrences, contexts, substitutions and matching, and the concept of a rewrite rule. Only simple examples of term rewriting sytems are given, just to illustrate the definitions. A preliminary discussion of equational reasoning and its semantics is given. As an application of rewriting, it is explained how complete term rewriting systems can sometimes be used to solve a word problem. The chapter concludes with an analysis of overlap between reduction rules, leading to the notion of a critical pair and to the Critical Pair Lemma.

A *term rewriting system* (TRS) is an abstract reduction system where the objects are *first-order terms*, and where the reduction relation is presented in a standard schematic format of so-called *reduction rules* or *rewrite rules*. This format makes use of the structure of first-order terms.

A typical example is the set of rewrite rules of Herbrand which was discussed in Section 0.3. Consider e.g. the rewrite rule

$$x + 0 \ \rightarrow \ x$$

It is schematic in two ways. First, arbitrary substitutions of a term for the variable x are allowed. This way we get e.g. the rewrite step

$$1 + 0 \ \rightarrow \ 1$$

as an instance of the rule $x + 0 \rightarrow x$, by substituting 1 for x. Secondly, a rewrite step thus obtained can be performed at arbitrary positions, somewhere inside a more complex term. In this example this means that any occurrence of $1 + 0$ may be replaced by 1. So we have for example

$$\neg(1 + 0) \vee y \Rightarrow 0 \quad \rightarrow \quad \neg 1 \vee y \Rightarrow 0$$

We say that rewriting is *closed under substitution* and can be performed in any *context*.

In this chapter we formally introduce the concept of a term rewriting system and its basic notions. We answer a number of elementary questions, and give

simple examples of TRSs and many exercises. The purpose is to let the reader get acquainted with the subject matter of this book. Moreover, several topics that are covered in the later chapters are introduced here and explained, with reference to the places where the material is further investigated and expanded.

Some more extended examples, which are also meant to serve as running examples in the later chapters, will be treated separately in Chapter 3. One specific TRS will get extra attention there (in Section 3.2): the system of combinatory logic (CL). It may be considered as *the* paradigmatic example of a TRS. As a matter of fact, the theory of term rewriting systems largely originates in the theory of CL and the closely related λ-calculus. The λ-calculus was proposed in the 1930s by A. Church [1936, 1941] as a basis for the notion of a computable function. In Terese [2003] the λ-calculus is studied in Chapter 10.

Because of the presence of bound variables in its terms, the λ-calculus does not fit in the format of first-order term rewriting. It is an example of a so-called *higher-order* term rewriting system. In contrast, term rewriting systems without bound variables are called *first-order* term rewriting systems. A note on terminology is in order here. There is the obvious implication that first- and higher-order term rewriting systems are two branches of a, broader, general concept of term rewriting system. This is how one could interpret the title of this book. However, in most of the literature of the last two decades the notion of term rewriting system was understood in the restricted first-order sense. Higher-order systems are then considered an extension of the standard format. In this chapter, where we will restrict ourselves to first-order systems anyway, we will adopt this convention. Chapter 11 of Terese [2003] is devoted to the general theory of higher-order rewriting systems.

2.1. Terms

A *term rewriting system* (TRS) consists of terms and rules for rewriting (also called reducing) these terms. So we first need the terms. Briefly, they will be just the terms over a given first-order signature, as in first-order predicate logic. Here is a formal definition.

2.1.1. Definition. A *signature* Σ consists of a non-empty set of *function symbols* or *operator symbols* $F, G, \ldots$, each equipped with a fixed *arity*. The arity of a function symbol F is a natural number, indicating the number of arguments it is supposed to have. So we (may) have unary, binary, ternary, etc., function symbols. Also *nullary* (0-ary) functions are allowed: these are also called *constant symbols* or just *constants*.

Terms are strings of symbols from an *alphabet*, consisting of the signature and a countably infinite set *Var* of *variables*. The set of variables is assumed

to be disjoint from the function symbols in the signature Σ. Variables will be denoted by x, y, z, x', y', etc., if needed with indices: $x_0, x_1, x_2, x_3, \ldots$.

For the function symbols sometimes more suggestive names will be employed. For example c or 0 for a constant, *Plus* or $+$ for a binary function symbol, etc. In the following definition of terms the function symbols are used in prefix notation. Also postfix and infix notation, or even other formats, may be used when appropriate – as long as they can be unequivocally brought back to conform with Definition 2.1.2.

2.1.2. DEFINITION. The set of *terms* over Σ is indicated as $Ter(\Sigma)$ and is defined inductively:

(i) $x \in Ter(\Sigma)$ for every $x \in Var$.

(ii) If F is an n-ary function symbol ($n \geq 0$) and $t_1, \ldots, t_n \in Ter(\Sigma)$, then $F(t_1, \ldots, t_n) \in Ter(\Sigma)$. By this notation it is understood (case $n = 0$) that the constant symbols of Σ are in $Ter(\Sigma)$ – we write c instead of $c(\,)$.

The terms t_i are called the *arguments* of the term $F(t_1, \ldots, t_n)$, and the symbol F the *head symbol* or *root*. Notation $F \equiv root(t)$.

Terms not containing a variable are called *closed* terms (also *ground* terms), and $Ter_0(\Sigma)$ is the set of closed terms. Terms in which no variable occurs more than once are called *linear*.

By $Var(t)$ we denote the set of variables that occur in t. So t is a closed term if $Var(t) = \varnothing$. If V is a set of variables, then the notation $Ter(\Sigma, V)$ is often used to denote the set of terms t with $Var(t) \subseteq V$. According to this convention we have $Ter_0(\Sigma) = Ter(\Sigma, \varnothing)$.

The *length* of the term t, denoted by $|t|$, is defined as the number of occurrences of function symbols and variables in t. So we have the inductive definition $|x| = |c| = 1$, for a variable x and a constant c, and $|F(t_1, \ldots, t_n)| = |t_1| + \cdots + |t_n| + 1$, for a function symbol F with arity $n \geq 1$.

Identity of terms is called *syntactic identity* and is denoted by $\equiv$. Terms will be the objects of term rewriting systems, where term rewriting systems are instances of abstract rewrite systems. So this notation is in conformity with the use of the symbol $\equiv$ in Chapter 1.

2.1.1. Contexts, occurrences

We will introduce several notational devices by which we obtain some flexibility in describing terms and operations on terms. One such device is the notion of *context*. A context can be viewed as an incomplete term, which may contain several empty places, or *holes*.

Formally a context can be defined as a term containing zero, one or more occurrences of a special constant symbol $\Box$, denoting holes, i.e., a term over the extended signature $\Sigma \cup \{\Box\}$. If C is a context containing exactly n holes,

and $t_1, \ldots, t_n$ are terms, then $C[t_1, \ldots, t_n]$ denotes the result of replacing the holes of C from left to right by $t_1, \ldots, t_n$. If $t \in Ter(\Sigma)$ can be written as $t \equiv C[t_1, \ldots, t_n]$, then the context C will also be called a *prefix* of t. If $t \equiv D[C[t_1, \ldots, t_n]]$ for some prefix D, then C is a *subcontext* of t, sometimes also called a *slice*.

An important special case is when there is exactly one occurrence of $\Box$ in C. Then C is called a *one-hole context*, also denoted by $C[\,]$; the notation $C[\,]$ is used exclusively for one-hole contexts. If $t \in Ter(\Sigma)$ can be written as $t \equiv C[s]$, then the term s is said to be a *subterm* of t, notation $s \subseteq t$. Since $\Box$ is itself a context, the *trivial context*, we also have $t \subseteq t$. Other subterms s of t than t itself are called *proper* subterms of t, notation $s \subset t$.

Since contexts have been introduced as terms over an extended signature, they automatically inherit the inductive definition of terms, and thereby the facility of giving recursive definitions and proofs by induction on contexts.

2.1.3. Exercise. (i) Give a direct inductive definition of one-hole contexts.

(ii) Define $C[t]$, the result of replacing the hole in $C[\,]$ by t, by recursion on $C[\,]$.

Note that the same term s may occur more than once as a subterm in a term t. Often it is important to distinguish between different occurrences of a subterm (or symbol) in a term. In these cases the notion of subterm is not precise enough. There are several ways to formally define the notion of subterm *occurrence*.

One obvious way, which we shall adopt as working definition, uses the prefix of the occurrence: the ordered pair $\langle s \mid C[\,]\rangle$ uniquely determines the occurrence of s in $C[s]$ with prefix $C[\,]$. Analogously, the pair $\langle f \mid C[\,]\rangle$ determines a unique occurrence of the function symbol f in the term $C[f(t_1, \ldots, t_n)]$, namely the head symbol of the occurrence $\langle f(t_1, \ldots, t_n) \mid C[\,]\rangle$. An informal method of identifying occurrences within a term is by typographic means such as underlining or using bold-face type.

2.1.4. Example. Let $\Sigma = \{A, M, S, 0\}$, with arities $2, 2, 1$ and 0, respectively. Then we have the following.

- $A(M(x, y), y)$ is a non-linear term, with two occurrences of the variable y.
- $A(M(x, y), z)$ is a linear term.
- $A(M(S(0), 0), S(0))$ is a ground term.
- $A(M(\Box), \Box), S(\Box))$ is a prefix of $A(M(S(0), 0), S(0))$.
- $S(0)$ is a subterm of $A(M(S(0), 0), S(0))$, having two occurrences. The corresponding prefixes are $A(M(\Box, 0), S(0))$ and $A(M(S(0), 0), \Box)$. So in our formal notation the occurrences are represented as $\langle S(0) \mid A(M(\Box, 0), S(0))\rangle$ and $\langle S(0) \mid A(M(S(0), 0), \Box)\rangle$, respectively.

For the relative position of two occurrences in a given term t there are two possibilities. Either one occurrence *contains* the other, or the two occurrences are completely separate. In the latter case we call them *disjoint*. This rather intuitive description is made precise in the following formal definition. The two indicated possibilities are exhaustive (see Exercise 2.1.11).

2.1.5. DEFINITION. Consider two occurrences $\langle s \mid C[\,]\rangle$ and $\langle s' \mid C'[\,]\rangle$ in a term t.

(i) We say that $\langle s' \mid C'[\,]\rangle$ is *contained* in $\langle s \mid C[\,]\rangle$, notation $\langle s' \mid C'[\,]\rangle \leq \langle s \mid C[\,]\rangle$, if for some context $D[\,]$ we have $C'[\,] \equiv C[D[\,]]$.

(ii) We say that $\langle s \mid C[\,]\rangle$ and $\langle s' \mid C'[\,]\rangle$ are *disjoint* if for some two-hole context E either $C[\,] \equiv E[\Box, s']$ and $C'[\,] \equiv E[s, \Box]$, or, in the reverse order, $C[\,] \equiv E[s', \Box]$ and $C'[\,] \equiv E[\Box, s]$.

Note that by this definition two occurrences of subterms in t are disjoint if, when viewed as strings of symbols occurring in t, they have no symbol occurrence in common. In case the occurrence $\langle s' \mid C'[\,]\rangle$ is contained in $\langle s \mid C[\,]\rangle$, we of course expect that $s' \subseteq s$. Indeed, from $C[s] \equiv t \equiv C'[s'] \equiv C[D[s']]$ it follows that $s \equiv D[s']$.

2.1.6. DISCUSSION. Now we have been very precise about occurrences. But, as a matter of fact, we do not always want to be too tight about it. So in cases where no confusion is likely, for example when s occurs only once in t, or when one particular occurrence of s is considered, we will often use the subterm to denote the occurrence, and just write: 'the occurrence s ...'. As a consequence, when $s \subseteq t$, we may also write $s \leq t$, when we want s, t to be considered as occurrences.

2.1.2. Term trees

A term can be represented as a finite labelled, non-commutative tree according to the following inductive definition.

(i) A variable or a constant is represented by the tree consisting of one single node, labelled by the variable or constant itself.

(ii) The term $F(t_1, \ldots, t_n)$ is represented by the tree of which the root node has as its label the symbol F, and which has as its immediate subtrees, in order from left to right, the trees representing $t_1, \ldots, t_n$.

The tree representing t will be called the *term tree* of t. See Figures 2.1 and 2.2.

2.1.7. DEFINITION. The *depth* of the term t, denoted by $depth(t)$, is defined as the number of nodes on a branch of the term tree of t of maximal length. So we have the inductive definition

(i) $depth(x) = depth(c) = 1$,

(ii) $depth(F(t_1, \ldots, t_n)) = max\{depth(t_1), \ldots, depth(t_n)\} + 1$, for $n \geq 1$.

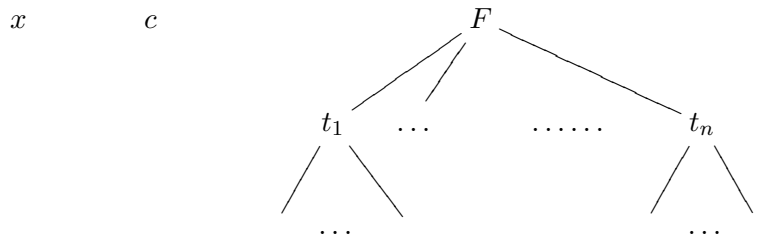


Figure 2.1: Term trees of x, c and $F(t_1, \ldots, t_n)$

Nodes in a term tree are usually called *positions*. This terminology is then transferred to the corresponding term, so that we also speak of positions in terms. A natural way to code a node in a finitely branching ordered tree is by a finite, possibly empty, sequence of positive integers, indicating the path from the root of the term tree to the intended position (see Figure 2.2). Such sequences will be used to denote the corresponding positions.

2.1.8. NOTATION. The notation $\langle k_1, \ldots, k_n \rangle$ is used for sequences (see also Definition A.1.4). The concatenation of the sequences p and q is denoted by $p \cdot q$. For convenience, we use the same notation for the extension (at front or tail) of the sequence p with just one element i. So we have

- $\langle k_1, \ldots, k_n \rangle \cdot \langle k_{n+1}, \ldots, k_{n+m} \rangle = \langle k_1, \ldots, k_{n+m} \rangle$,
- $i \cdot \langle k_1, \ldots, k_n \rangle = \langle i, k_1, \ldots, k_n \rangle$,
- $\langle k_1, \ldots, k_n \rangle \cdot i = \langle k_1, \ldots, k_n, i \rangle$.

We write $p \leq q$ if q extends p, that is, if $q = p \cdot p'$ for some sequence p'; and we write $p \mid\mid q$ if neither $p \leq q$, nor $q \leq p$. Note that $p \leq q$ if and only if q is in the subtree with p as root.

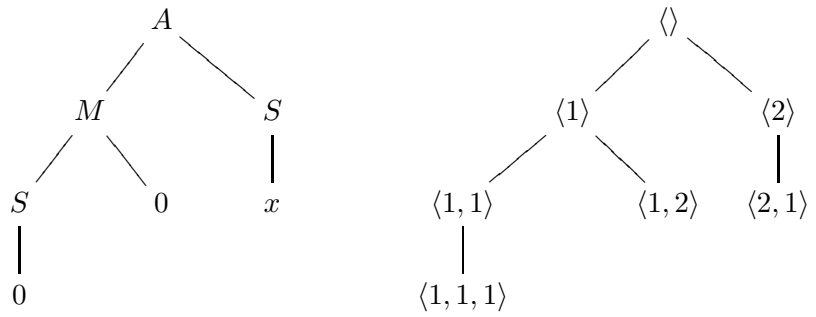


Figure 2.2: Term tree of $A(M(S(0), 0), S(x))$, with corresponding positions

A position in (the term tree of) a term t determines both a prefix of t, namely the prefix with its (only) hole at position p, and (an occurrence of)

the subterm with that prefix. The prefix of t that corresponds to p is denoted by $t[\ ]_p$, the subterm occurring at position p by $t|_p$. We will admit some overloading of notation and use $t|_p$ also for the occurrence $\langle t|_p \mid t[\ ]_p\rangle$. We call it the *occurrence at p in t*. If $t[\ ]_p$ is the context $C[\ ]$, then by $t[s]_p$ we (naturally) denote the term $C[s]$. Notation is liberalized even further, by allowing a prefix to take the place of a position in the notation $t|_{C[\ ]}$ for s if $t \equiv C[s]$.

Note that if we are interested in occurrences of *symbols* (i.e., variables and function symbols), instead of subterms, positions can be employed in the same way. If p is a position in a term t, then p also determines the symbol occurring in t at (the node) p. This symbol is sometimes denoted by $t(p)$.

2.1.9. Example. For an illustration of the concepts discussed here see Figure 2.2. We have for $t \equiv A(M(S(0),0),S(x))$ for example that $t|_{\langle 1,1\rangle} \equiv S(0)$ and $t[\ ]_{\langle 1,1\rangle} \equiv A(M(\Box,0),S(x))$.

2.1.10. Exercise. (i) Convince yourself of the correctness of the following inductive definition of the subterm of t at position p. (Defined only for positions p that actually occur in t.)

$$\begin{aligned} t|_{\langle\rangle} &\equiv t \\ F(t_1,\ldots,t_n)|_{i\cdot p} &\equiv t_i|_p \end{aligned}$$

(ii) Analogously, give an inductive definition of the context $t[\ ]_p$.

2.1.11. Exercise. (i) Verify that for occurrences we have $t|_p \leq t|_q$ if and only if $q \leq p$.

(ii) Verify that $t|_p$ and $t|_q$ are disjoint if and only if $q \parallel p$.

(iii) Conclude that for two occurrences in a given term there are only two possibilities: one contains the other or they are disjoint.

2.1.12. Exercise. Prove the following identities.

(i) $t|_{p\cdot q} \equiv t|_p|_q$.
(ii) $t[s]_p|_{p\cdot q} \equiv s|_q$.
(iii) $t[s]_{p\cdot q}|_p \equiv t|_p[s]_q$.
(iv) $t[s]_{p\cdot q}[r]_p \equiv t[r]_p$.
(v) If $p \parallel q$, then $t[s]_p|_q \equiv t|_q$.

2.1.3. Substitution

Substitution is the operation of filling in terms for variables. In the following assume a signature Σ to be given.

2.1.13. Definition. A *substitution* is a map σ: $Ter(\Sigma) \to Ter(\Sigma)$ which satisfies

$$\sigma(F(t_1,\ldots,t_n)) \equiv F(\sigma(t_1),\ldots,\sigma(t_n))$$

for every n-ary function symbol F ($n \geq 0$); in particular, $\sigma(F) \equiv F$ if F is a constant symbol. We normally write t^σ (or $t\sigma$) instead of $\sigma(t)$.

The substitution σ is determined by its restriction to the set of variables. Therefore it can equivalently be defined as a map $\sigma\colon\ \mathit{Var} \to \mathit{Ter}(\Sigma)$. This approach is quite common. Actual substitutions will mostly be given in this way. The two definitions of substitution are not distinguished and will indiscriminately be denoted by σ. As a matter of fact, t^σ only depends on the values of σ for variables that actually occur in t.

Observe that, since we regard contexts as terms over an extended signature, substitution is automatically also defined for contexts.

It is customary, especially in the literature on logic programming, to require of a substitution that it affects only finitely many variables, and acts on the other ones as the identity. One then defines $\mathit{Dom}(\sigma) = \{x \mid \sigma(x) \not\equiv x\}$ as the *domain* of σ – even though substitutions are supposed to be total functions. We will not make this finite domain restriction. However, for substitutions with a finite 'domain' an explicit notation can be given, and several such notations are quite common in the literature.

2.1.14. Notation. Already in the field of the λ-calculus, the natural home resort of substitution, a great variety of such explicit notations is used. In this book we adopt the notation $[x_1, \ldots, x_n := s_1, \ldots, s_n]$ for the substitution σ with $\sigma(x_i) \equiv s_i$ for $1 \leq i \leq n$ and $\sigma(y) \equiv y$ for all $y \not\equiv x_i$, $1 \leq i \leq n$. As shorthand, vector notation may be employed: $[\vec{x} := \vec{s}]$.

Another notation that is sometimes encountered, in particular in the logic programming literature, represents the substitution as a finite set of assignments to variables: $\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$ for the substitution $[x_1, \ldots, x_n := s_1, \ldots, s_n]$. A frequent variant of this notation is $\{x_1/s_1, \ldots, x_n/s_n\}$. However, it is equally common to write $\{s_1/x_1, \ldots, s_n/x_n\}$.

When $V \subseteq \mathit{Var}$ we write $\sigma|_V$ for the substitution defined by $\sigma|_V(x) = \sigma(x)$ for $x \in V$ and $\sigma|_V(x) = x$ for $x \notin V$. Slightly confusingly, $\sigma|_V$ is sometimes called the *restriction of σ to V*. Note that we have $\mathit{Dom}(\sigma|_V) \subseteq V$, but not necessarily $\mathit{Dom}(\sigma|_V) = V$, something one might have expected.

If σ_1, σ_2 are substitutions with disjoint finite domains, then $\sigma_1 \cup \sigma_2$ denotes the substitution with $\mathit{Dom}(\sigma_1 \cup \sigma_2) = \mathit{Dom}(\sigma_1) \cup \mathit{Dom}(\sigma_2)$ defined by $(\sigma_1 \cup \sigma_2)(x) = \sigma_i(x)$ if $x \in \mathit{Dom}(\sigma_i)$, for $i = 1, 2$. Note that the logic programming notation suits this definition nicely.

The composition of two substitutions is again a substitution. Since substitutions are written postfixed, it is also convenient to write the composition of σ and τ as $\tau\sigma$ ('first τ then σ'). The substitution $\tau\sigma$ is called a *refinement* of τ, and we say that τ is *more general* than $\tau\sigma$.

A substitution σ that replaces distinct variables by distinct variables (i.e. σ is injective and x^σ is a variable for every x) is called a *renaming*. If $\sigma|_{\mathit{Var}(t)}$ is a renaming, then σ is called a *renaming for t*.

2.1.15. Exercise. (i) Show that composition of substitutions is associative, but not commutative.

(ii) Show that for ground t we always have $t^\sigma \equiv t$.

(iii) Show that $t^\sigma \equiv t^\tau$, if and only if σ and τ coincide on the variables in t.

2.1.4. Matching, subsumption, unification

Substitution induces a quasi-order on terms, the so-called *subsumption* order. See Definition 2.1.16 and Exercise 2.1.17.

2.1.16. Definition (subsumption). Let $s \equiv t^\sigma$. Then s is called a *(substitution) instance* of t, and we say that t *subsumes* s, written as $s \preceq t$. The term t is also said to be *matched* with its instance s by the substitution σ. Moreover,

(i) if σ is not a renaming for t, then s is called a *proper* instance of t, notation $s \prec t$,

(ii) if σ is a renaming for t, then we say that s is a *variant* of t, and write $s \cong t$.

2.1.17. Exercise (subsumption order). (i) Show that the subsumption order is a quasi-order on terms, i.e., the relation $\preceq$ is reflexive and transitive.

(ii) Show that $t \cong s \;\Leftrightarrow\; t \preceq s \;\wedge\; s \preceq t$.

(iii) Show that $t \prec s \;\Leftrightarrow\; t \preceq s \;\wedge\; t \not\succeq s$.

A matching substitution can be computed by the matching algorithm of Table 2.1, acting on finite sets of would-be matchings $t \mapsto s$. The algorithm proceeds by performing (any one of) the given transformation rules, as long as they are applicable. To determine a substitution that matches t with s, the algorithm should be started on the singleton set $\{t \mapsto s\}$. Note that each line of the algorithm is of the form $\{\mathit{matching}\} \cup S \Longrightarrow \cdots$, with S a finite set of matchings. Here it is understood that $\mathit{matching} \notin S$.

$$
\begin{aligned}
\{F(t_1,\ldots,t_n) \mapsto F(s_1,\ldots,s_n)\} \cup S &\Longrightarrow \{t_1 \mapsto s_1,\ldots,t_n \mapsto s_n\} \cup S\\
\{F(t_1,\ldots,t_n) \mapsto G(s_1,\ldots,s_m)\} \cup S &\Longrightarrow \textbf{fail}, \text{ if } F \not\equiv G\\
\{F(t_1,\ldots,t_n) \mapsto x\} \cup S &\Longrightarrow \textbf{fail}\\
\{x \mapsto s_1\} \cup S &\Longrightarrow \textbf{fail}, \text{ if } x \mapsto s_2 \in S, \text{ for some } s_2 \text{ with } s_1 \not\equiv s_2
\end{aligned}
$$

Table 2.1: Matching algorithm

It is not hard to verify the correctness of the matching algorithm. That is, when we start on the singleton set $\{t \mapsto s\}$, then there are two possibilities.

– If a substitution matching t with s exists, then the algorithm terminates with that substitution in 'logic programming notation'.

– When no matching substitution exists, then the algorithm terminates with **fail**.

There is even more structure to the subsumption order: for any two terms there exists a least general term of which both are instances, i.e. a supremum with respect to $\preceq$. So the set of terms with the subsumption order form an upper semi-lattice. By elementary algebraic reasoning it then also follows that if two terms have a common instance, they have a *most general* common instance (mgci), i.e. an infimum with respect to $\preceq$. In Exercise 2.1.18 the reader is invited to establish these facts on the basis of an algorithm for computing the supremum.

2.1.18. Exercise (most general common instance). (i) Devise an algorithm for computing the supremum of two terms t and s.

(ii) Prove that the strict subsumption order is an inverse well-order, i.e., no infinite ascending chains $t_1 \prec t_2 \prec \cdots$ exist.

(iii) Prove, using (i) and (ii), that two terms with at least one common instance have an mgci.

(iv) Show that an mgci of s, t is unique up to renaming.

The subsumption order on terms can be extended to substitutions.

2.1.19. Definition. For substitutions we define $\sigma \preceq \tau \Leftrightarrow (\exists \rho)\ \sigma = \tau\rho$. If $\sigma \preceq \tau$ we say that τ is *more general* than σ.

2.1.20. Exercise. (i) Show that if $\sigma \preceq \tau$ then $t^\sigma \preceq t^\tau$ for every term t.

(ii) Show that the inverse implication is not true in general.

If $s^\sigma \equiv t^\sigma$, then the substitution σ is called a *unifier* for t and s (by which t and s are *unified*). If t and s are unifiable, then there is, again, a *most general unifier* (mgu), that is, a unifier that is minimal with respect to the ordering $\preceq$ on substitutions. Just as in the case of mgci's, the mgu is unique up to renaming. Exercise 2.1.21 covers the relationship between mgu's and mgci's.

2.1.21. Exercise (most general unifiers). (i) Assume $Var(s) \cap Var(t) = \varnothing$. Show that u is an mgci of s, t if and only if $u \equiv s^\sigma$ for some mgu σ of s, t.

(ii) Given an algorithm for mgu's, show how to compute the mgci of terms t, s, if it exists.

(iii) Given an algorithm for mgci's, show how to compute the mgu of terms t, s, if it exists.

We now briefly indicate a direct algorithm for computing mgu's, due to Martelli and Montanari [1982]. The Martelli–Montanari algorithm is often treated in textbooks on logic programming and resolution, for example Apt [1997], Doets [1994].

The algorithm, given in Table 2.2, acts on finite sets of equations. To compute an mgu for s and t, it should be run with input $\{s = t\}$. Each

line of the algorithm is of the form $\{equation\} \cup E \Longrightarrow \cdots$, with E a finite set of equations. Just as in the matching algorithm, it is understood that $equation \notin E$. We explain the notation $E^{[x:=t]}$. If e is an equation $s_0 = s_1$, let $e^{[x:=t]}$ denote the equation $s_0^{[x:=t]} = s_1^{[x:=t]}$. Then $E^{[x:=t]}$ is the set of equations $e^{[x:=t]}$ with e in E.

$$
\begin{array}{rcll}
\{F(t_1,\ldots,t_n) = F(s_1,\ldots,s_n)\} \cup E & \Longrightarrow & \{t_1 = s_1,\ldots,t_n = s_n\} \cup E & \\
\{F(t_1,\ldots,t_n) = G(s_1,\ldots,s_m)\} \cup E & \Longrightarrow & \textbf{fail}, & \text{if } F \not\equiv G \\
\{x = x\} \cup E & \Longrightarrow & E & \\
\{F(t_1,\ldots,t_n) = x\} \cup E & \Longrightarrow & \{x = F(t_1,\ldots,t_n)\} \cup E & \\
\{x = F(t_1,\ldots,t_n)\} \cup E & \Longrightarrow & \textbf{fail}, & \text{if } x \in \mathit{Var}(F(t_1,\ldots,t_n)) \\
\{x = t\} \cup E & \Longrightarrow & \{x = t\} \cup E^{[x:=t]}, & \text{if } x \notin \mathit{Var}(t), x \in \mathit{Var}(E)
\end{array}
$$

Table 2.2: Martelli–Montanari unification algorithm

Note that matching can be viewed as a special case of unification. If $s \preceq t$, with matching substitution ρ, then ρ is also an mgu of s, t (and the mgci is s itself).

The issue of most general unifiers is fundamental for the important branch of the term rewriting discipline, concerned with *narrowing*. This a method for finding solutions for an equation $t = s$, given a set of equations E.

2.1.22. Exercise. Determine whether the following pairs of terms can be unified, and if so, give the most general unifier.

(i) $H(x, H(x, F(y)))$ and $H(H(G(y), y), z)$,
(ii) $H(H(x, y), x)$ and $H(H(F(y), z), G(z))$,
(iii) $H(H(G(x), F(y)), F(z))$ and $H(H(G(z), F(F(x))), F(y))$.

2.2. Definition of a TRS

After this exposition on terms we continue towards the definition of term rewriting systems. What is still missing is the second ingredient: the reduction rules.

2.2.1. Definition. A *reduction rule* (or rewrite rule) for a signature Σ is a pair $\langle l, r\rangle$ of terms of $\mathit{Ter}(\Sigma)$. It will be written as $l \to r$. Often a reduction rule will get a name, e.g. ρ, and we write $\rho : l \to r$. Two restrictions on reduction rules will be imposed:

(i) the left-hand side l is not a variable,
(ii) every variable occurring in the right-hand side r occurs in l as well.

A reduction rule $\rho : l \to r$ can be viewed as a scheme. An *instance* of ρ is obtained by applying a substitution σ. The result is an *atomic* reduction step $l^\sigma \to_\rho r^\sigma$. The left-hand side l^σ is called a *redex* (from *red*ucible *ex*pression), more precisely a ρ-redex. The right-hand side r^σ is called its *contractum*.

2.2.2. EXAMPLE. Consider a rewrite rule $\rho : F(G(x), y) \to F(x, x)$. Then a substitution σ, with $\sigma(x) = 0$ and $\sigma(y) = G(x)$, yields the atomic reduction step

$$F(G(0), G(x)) \to_\rho F(0, 0)$$

with redex $F(G(0), G(x))$ and contractum $F(0, 0)$.

In accordance with what was said in Discussion 2.1.6 the words redex and contractum will also be used for occurrences of a redex and a contractum. Replacing a redex l^σ by its contractum is sometimes called *contraction*, instead of reduction or rewriting, in order to stress the active role that is played by the redex.

2.2.3. REMARK. A redex r is somehow tied to the reduction rule according to which it is a redex. It may happen, however, that r can be considered both as a ρ_1-redex and as a ρ_2-redex, with respect to different rules ρ_1, ρ_2 (see Example 2.2.9). The notion of *redex* will be assumed to account for this type of information in an implicit way, so that actually there are two redexes r in this case: one with respect to ρ_1, and one with respect to ρ_2. For the moment we leave it at this informal level. Thereby we adhere to the traditional use of the word redex of e.g. Curry and Feys [1958] and Barendregt [1984].

Given a term, it may contain one or more occurrences of redexes. A reduction step consists of contracting one of these, i.e. replacing the redex by its contractum.

2.2.4. DEFINITION. A *reduction step* (or rewrite step) according to the reduction rule $\rho : l \to r$ consists of contracting a redex within an arbitrary context:

$$C[l^\sigma] \to_\rho C[r^\sigma]$$

We call $\to_\rho$ the *one-step reduction relation* generated by ρ.

2.2.5. EXAMPLE. The contraction indicated in Example 2.2.2 above gives rise e.g. to the reduction step

$$F(z, G(F(G(0), G(x)))) \to_\rho F(z, G(F(0, 0)))$$

Here the context is $F(z, G(\Box))$.

2.2.6. DISCUSSION. The two restrictions on reduction rules, although they may at first sight appear somewhat arbitrary, become quite natural if one looks upon a reduction rule as a rule of computation. Compare a natural restriction on what one normally would consider as a sound way to define a function f. A definition introducing a new variable, like $f(x) = x + y$, would not be allowed.

There are also more technical reasons. Notice that a rule violating restriction (i), like $\rho_1 : x \to c$, would turn every term into a redex; there are no normal forms in the presence of such a reduction rule. A rule violating condition (ii), like $\rho_2 : c \to x$, would be explosive in another manner: since any term can be substituted in the

right-hand side of ρ_2 without affecting the left-hand side, there would be no bound on the size of the contractum in terms of the redex it comes from. Normally there is no use for rules that exhibit such uncontrollable behaviour. By the way, both the rules ρ_1 and ρ_2 would have the trivializing effect that all terms become convertible to c.

Notwithstanding the above considerations, the restrictions on rewrite rules are not a priori necessary and some authors prefer not to have them. Moreover, in the setting of *conditional* term rewriting systems (see Section 3.4) the restrictions lose some of their motivation: conditions added to a rewrite rule provide the opportunity for compensating the indicated lack of control. We will now give the general definition of a TRS. A similar system, but with reduction rules that violate one or both restrictions in Definition 2.2.1, may then be called a *pseudo-TRS*.

2.2.7. Definition. (i) A *term rewriting system* is a pair $\mathcal{R} = (\Sigma, R)$ of a signature Σ and a set of reduction rules R for Σ.

(ii) The *one-step reduction* relation of $\mathcal{R}$, denoted by $\to$ (or by $\to_R$, when we want to be more specific), is defined as the union $\bigcup\{\to_\rho \mid \rho \in R\}$. So we have $t \to_R s$ when $t \to_\rho s$ for one of the reduction rules $\rho \in R$.

If $\mathcal{R} = (\Sigma, R)$, then we will instead of $Ter(\Sigma)$ also write $Ter(\mathcal{R})$. Likewise $Ter_0(\mathcal{R})$. Often we will specify a TRS just by its set of reduction rules R. Then the signature Σ is tacitly understood to consist solely of the function symbols that are used in R.

We allow some freedom in the use of subscripts for $\to$. In practice, the index ρ (or R) will mostly be suppressed. But sometimes we want to be even more specific, by giving some indication of the contracted redex (occurrence) in the subscript. So we may write for example $t \to_{C[\,]} s$, if the contracted redex occurs in t with prefix $C[\,]$, or $t \to_{\rho, C[\,]} s$, if we still want to explicitly mention the reduction rule used.

2.2.8. Example. Consider the TRS with reduction rules

$$\begin{array}{lrcl} \rho_1: & F(a,x) & \to & G(x,x) \\ \rho_2: & b & \to & F(b,b) \end{array}$$

- The substitution $[x := b]$ yields the atomic reduction step $F(a,b) \to_{\rho_1} G(b,b)$.
- A corresponding one-step reduction is $G(F(a,b),b) \to_{F(a,b)} G(G(b,b),b)$.
- Another one-step reduction is $G(F(a,b),b) \to_b G(F(a,b),F(b,b))$.
- In the second reduction, since there are two occurrences of b, it would make sense to be more specific on the contracted redex by writing for example $G(F(a,b),b) \to_{\langle b \,|\, G(F(a,b),\Box)\rangle} G(F(a,b),F(b,b))$.

In the above examples most of the information that is rendered by the index is actually redundant, in the sense that it can be reconstructed if the beginning and end terms of the reduction step are given. This will not always be the case. We give three more examples.

2.2.9. EXAMPLES. (i) If a TRS has the reduction rule

$$a \to a$$

then in a reduction step $F(a,a) \to F(a,a)$ it is not clear which redex a is contracted. We may want to distinguish between steps $F(a,a) \to_{F(\Box,a)} F(a,a)$ and $F(a,a) \to_{F(a,\Box)} F(a,a)$.

(ii) A similar example can be constructed with a reduction rule

$$I(x) \to x$$

We have $I(I(x)) \to I(x)$, but then it is not yet clear whether the step is actually $I(I(x)) \to_{I(I(x))} I(x)$ or $I(I(x)) \to_{I(x)} I(x)$.

(iii) Consider a TRS with rules

$$\begin{array}{lrcl} \rho_1: & F(a,x) & \to & G(x,x) \\ \rho_2: & F(x,a) & \to & G(x,x) \end{array}$$

and a reduction step $F(a,a) \to G(a,a)$. Now it is clear that the contracted redex is $F(a,a)$, but not which rule is applied. There are actually two steps: $F(a,a) \to_{\rho_1} G(a,a)$ and $F(a,a) \to_{\rho_2} G(a,a)$. It cannot be determined from the contracted redex which reduction rule was actually used. In accordance with Remark 2.2.3 we may distinguish a ρ_1-redex $F(a,a)$ and a ρ_2-redex $F(a,a)$.

The first two examples exhibit what are sometimes called *syntactic accidents*. The last example is a case of *overlapping* reduction rules. Overlap will be defined and analysed in Subsection 2.7.1.

2.3. Reduction

In a natural way a term rewriting system $\mathcal{R} = (\Sigma, R)$ gives rise to a corresponding abstract reduction system, namely $(Ter(\Sigma), \to)$. As a matter of fact, we will identify the two, with the effect that $\mathcal{R}$ is considered as an ARS by itself. Sometimes, when we want to be more specific about the rules being applied, we instead consider the TRS as the ARS with multiple reduction relations $(Ter(\Sigma), (\to_\rho)_{\rho \in R})$.

All ARS definitions and results from Chapter 1 carry over to the TRS world. We recapitulate only some of the more central notions.

Concatenating reduction steps, we have (possibly infinite) *reduction sequences*, $t_0 \to t_1 \to t_2 \to \cdots$, or *reductions* for short. If $t_0 \to \cdots \to t_n$ $(n \geq 0)$ we also write $t_0 \twoheadrightarrow t_n$, and t_n is called a *reduct* of t_0.

No reduction steps are possible from a term not containing any redex occurrence; such a term is a *normal form*. A term t is called *weakly normalizing* (WN) if it can be reduced to a normal form; t is called *strongly normalizing* (SN) if it has no infinite reductions; and t is called *confluent* (CR) if any two reducts of t have a common reduct. A TRS is WN, SN, CR, if all its terms are. A TRS is called *complete* if it is both SN and CR. If all ground

terms are WN, then the TRS is *ground*-WN; and likewise we have *ground*-SN, *ground*-CR.

Recall that the symbol '=' is reserved for the convertibility relation, i.e. the equivalence relation generated by $\to$. We have $s = t$ if and only if there exists a conversion sequence between s and t, i.e. a sequence $s \equiv s_0 \leftrightarrow s_1 \leftrightarrow \cdots \leftrightarrow s_n \equiv t$, $n \geq 0$.

If $s = t$ in a complete TRS, then there exists a unique normal form n such that $s \twoheadrightarrow n$ and $t \twoheadrightarrow n$.

2.3.1. EXAMPLE. Consider Σ as in Example 2.1.4. Let $\mathcal{N}^{AM} = (\Sigma, R)$ be the TRS (specifying the natural numbers with addition, multiplication, successor and zero) with reduction rules as given in Table 2.3.

ρ_1 :	$A(x,0)$	$\to$	x
ρ_2 :	$A(x,S(y))$	$\to$	$S(A(x,y))$
ρ_3 :	$M(x,0)$	$\to$	0
ρ_4 :	$M(x,S(y))$	$\to$	$A(M(x,y),x)$

Table 2.3: The TRS $\mathcal{N}^{AM}$

We have $M(S(S(0)),S(S(0))) \twoheadrightarrow S(S(S(S(0))))$, by the reduction in Table 2.4. Here in each step the underlined redex occurrence is rewritten. Note that this is not the only reduction from $M(S(S(0)),S(S(0)))$ to its normal form $S(S(S(S(0))))$.

$\underline{M(S(S(0)),S(S(0)))}$	$\to$	$\underline{A(M(S(S(0)),S(0)),S(S(0)))}$
	$\to$	$S(\underline{A(M(S(S(0)),S(0)),S(0))})$
	$\to$	$S(S(\underline{A(M(S(S(0)),S(0)),0)}))$
	$\to$	$S(S(\underline{M(S(S(0)),S(0))}))$
	$\to$	$S(S(A(\underline{M(S(S(0)),0)},S(S(0)))))$
	$\to$	$S(S(\underline{A(0,S(S(0)))}))$
	$\to$	$S(S(S(\underline{A(0,S(0))})))$
	$\to$	$S(S(S(S(\underline{A(0,0)}))))$
	$\to$	$S(S(S(S(0))))$

Table 2.4: A reduction

2.3.2. EXERCISE. (i) Reduce in the TRS $\mathcal{N}^{AM}$ of Example 2.3.1 the term $M(S(S(S(0))),S(S(0)))$ to normal form.

(ii) Determine the reduction graph of $M(S(0),A(0,S(0)))$.

The following properties of reduction rules will turn out to be important.

2.3.3. DEFINITION. (i) A reduction rule is called *left-linear*, if its left-hand side is a linear term. It is called *right-linear* if its right-hand side is a linear term.

(ii) A reduction rule is called *non-duplicating*, if no variable has more occurrences in the right-hand side than in the left-hand side. Otherwise it is called *duplicating*.

(iii) A reduction rule is called *non-erasing*, if each variable that occurs in the left-hand side also occurs (at least once) in the right-hand side. Otherwise it is called *erasing*.

(iv) A reduction rule is called *collapsing*, if its right-hand side is a variable. Otherwise it is called *non-collapsing*. Also a reduction *step* may be called *collapsing*, if the reduction rule that is used is a collapsing rule.

A *TRS* is called left-linear, non-duplicating, non-collapsing or non-erasing if all of its reduction rules are.

2.3.4. EXERCISE. (i) Assume that in a term t in a left-linear TRS we have occurrences $\langle r \mid t[\,]_p\rangle$ of a redex and $\langle x \mid t[\,]_q\rangle$ of a variable. Show that $t[s]_q|_p$ is again a redex. Show that this does not need to be true in a TRS that is not left-linear.

(ii) How about $t^{[x:=s]}|_p$?

(iii) Prove that a non-duplicating TRS is SN, if in each rule the right-hand side is shorter than the left-hand side. Show that this need not be true for a TRS that has a duplicating rule.

(iv) Call a TRS *length-non-increasing* if $t \to u \Rightarrow |t| \geq |u|$. Show that a TRS is length-non-increasing if and only if it is non-duplicating and $|l| \geq |r|$ for each rule $l \to r$.

2.3.5. EXERCISE (substitution property). Let σ be an arbitrary substitution in a given TRS. Prove the following statements.

(i) $t \to s \;\Rightarrow\; t^\sigma \to s^\sigma$.
(ii) $t \twoheadrightarrow s \;\Rightarrow\; t^\sigma \twoheadrightarrow s^\sigma$.
(iii) $t = s \;\Rightarrow\; t^\sigma = s^\sigma$.

2.3.6. EXERCISE (context property). Let $C[\,]$ be an arbitrary one-hole context in a given TRS. Prove the following statements.

(i) $t \to s \;\Rightarrow\; C[t] \to C[s]$.
(ii) $t \twoheadrightarrow s \;\Rightarrow\; C[t] \twoheadrightarrow C[s]$.
(iii) $t = s \;\Rightarrow\; C[t] = C[s]$.

Generalize this exercise to arbitrary (multi-hole) contexts.

2.3.7. EXERCISE. (i) Prove $s_1 \to s_2 \;\Rightarrow\; t[x := s_1] \twoheadrightarrow t[x := s_2]$.

(ii) Let σ, σ' be substitutions such that $\forall x \in Var(t)\ \sigma(x) \twoheadrightarrow \sigma'(x)$. Show that $t^\sigma \twoheadrightarrow t^{\sigma'}$.

2.3.8. EXERCISE. Let $\mathcal{R} = (\Sigma, R)$ be a TRS and σ a substitution that assigns variables to variables. $\mathcal{R}^\sigma$ is defined as the TRS (Σ, R^σ), where R^σ is the result of

replacing each rule $t \to s$ in R by the rule $t^\sigma \to s^\sigma$. Notice that $\mathcal{R}^\sigma$ need not be left-linear, even if $\mathcal{R}$ is. ($\mathcal{R}^\sigma$ will then be called a *collapse* of $\mathcal{R}$.)

(i) Prove that if $\mathcal{R}$ is SN, then so is $\mathcal{R}^\sigma$.

(ii) Show that the converse of this implication does not always hold.

2.3.9. Exercise. Prove or disprove SN for the following TRSs.

(i) $\{g(x) \to f(x), f(x) \to x\}$.
(ii) $\{g(f(x)) \to f(f(g(x)))\}$.
(iii) $\{g(f(x)) \to f(f(g(g(x))))\}$.
(iv) $\{f(f(x)) \to f(g(f(x)))\}$.

2.3.10. Exercise (D. Plump, C. Grabmayer). Let $\mathcal{R}$ be a TRS with rewrite relation $\to$, which is WN. Consider the reduction-relation $\to_{\text{inn}}$ on the terms of $\mathcal{R}$, which is defined as follows: A step $s \to_{\text{inn}} t$ consists in replacing an innermost redex r of s by a normal form n of r such that t is the result.

(i) Prove that $\to_{\text{inn}}$ is SN.

(ii) Prove that the reduction $\to_{\text{inn}}$ is also SN if we drop the assumption that the rewrite relation $\to$ is WN.

(iii) Actually, we can also drop the requirement "innermost". To be precise: Define the reduction-relation $\to_{\text{n}}$ from $\to$ as follows: A step $s \to_{\text{n}} t$ consists in replacing an arbitrary redex r in s by a normal form n of r such that t is the result. Prove that $\to_{\text{n}}$ is also SN.

(iv) Prove that $\to$ and $\to_{\text{inn}}$ have the same normal forms if and only if $\mathcal{R}$ is WN.

(v) Show that a TRS $\mathcal{R}$ is WN, if (and only if) every redex in $\mathcal{R}$ is WN. Observe the analogy with the statement in Exercise 5.2.2, which can be rendered as: A TRS $\mathcal{R}$ is SN, if (and only if) every redex in $\mathcal{R}$ is SN.

2.3.11. Exercise. Let $\mathcal{R}$ be a TRS with reduction relation $\to$. Define $\rhd = \ \to \cup \supset$. Here $\supset$ is the converse of the proper subterm relation. Prove that if $\to$ is SN, then so is $\rhd$.

2.3.12. Exercise. Find a one-rule TRS which is WN, but not SN.

2.4. Equational systems

In an equational specification equations occupy the position of the rewrite rules in a TRS. Equations are symmetric, and this is natural from an algebraic point of view. By replacing them by (directed) reduction rules the equations are transformed into rules of computation. A case in point is the TRS $\mathcal{N}^{AM}$ from Example 2.3.1. The equational variant $\mathcal{N}_{=}^{AM}$, given in Table 2.5, constitutes an equational specification of the set of natural numbers with addition and multiplication. The TRS $\mathcal{N}^{AM}$ is derived from $\mathcal{N}_{=}^{AM}$ by giving a specific orientation to the equations.

$e_1:$	$A(x,0)$	$=$	x
$e_2:$	$A(x,S(y))$	$=$	$S(A(x,y))$
$e_3:$	$M(x,0)$	$=$	0
$e_4:$	$M(x,S(y))$	$=$	$A(M(x,y),x)$

Table 2.5: The equational system $\mathcal{N}_{=}^{AM}$

Finding the 'right' orientation for the reduction rules that correspond to $\mathcal{N}_{=}^{AM}$ is obvious, given the recursive character of the equations. However, it will not always be a priori clear which direction the reductions should take. And in many cases *no* way of orienting the equations will yield a term rewriting system with satisfactory computational properties. For example, bluntly transforming an equation of the form $A(x,y) = A(y,x)$, expressing commutativity of addition, into a rewrite rule would result in non-termination.

Why would one want reduction rules instead of equations? A typical algebraic question that can sometimes be solved using term rewriting techniques is what is traditionally known as the *word problem*: deciding whether two terms are provably equal, given a set of equations. The word problem will be addressed first briefly in Section 2.6 in this chapter, and then much more extensively in Chapter 7. There may be other reasons for associating a term rewriting system to an equational theory. For example in functional programming the primary interest is in computing with equations, rather than in mere algebraic manipulation.

Let us now be more specific about the formal notion of an equational system, or *equational specification*.

2.4.1. Definition. An equational specification is a pair $\mathcal{E} = (\Sigma, E)$ where Σ is a signature as in Section 2.1 and where E is a set of equations $s = t$ between terms $s, t \in Ter(\Sigma)$.

Comparing this definition with the definition of a term rewriting system (Definition 2.2.7), the only difference is that we now have equations instead of reduction rules. Since there are no restrictions on the pairs of terms that constitute an equation, the natural correspondence is with the concept of pseudo-TRS.[1] The essential difference with a reduction rule is of course that the order of the terms in an equation is considered irrelevant. Some natural transitions between equational specifications and term rewriting systems are given in the following definition.

[1]Recall that a pseudo-TRS was defined as a TRS with reduction rules that do not necessarily satisfy the restrictions (i) and (ii) of Definition 2.2.1.

2.4.2. DEFINITION. (i) Given an equational specification $\mathcal{E} = (\Sigma, E)$, we denote by $\mathcal{E}^{\leftrightarrow}$ the pseudo-TRS $(\Sigma, E^{\leftrightarrow})$, where the set of reduction rules is defined by $E^{\leftrightarrow} = \{s \to t \mid s = t \in E \text{ or } t = s \in E\}$.

(ii) Given a TRS $\mathcal{R} = (\Sigma, R)$, we denote by $\mathcal{R}^{=}$ the equational specification $(\Sigma, R^{=})$, where the set of equations is defined by $R^{=} = \{s = t \mid s \to t \in R\}$. The pseudo-TRS $(\mathcal{R}^{=})^{\leftrightarrow}$, the bi-directional variant of $\mathcal{R}$, we will simply denote by $\mathcal{R}^{\leftrightarrow}$.

(iii) A reduction sequence in $\mathcal{E}^{\leftrightarrow}$ is called a *conversion* according to (or *in*) the equational specification $\mathcal{E}$. We write $s =_E t$ if there exists a conversion from s to t in $\mathcal{E}$.

In Exercise 2.4.3 it will be established that the transitions described here leave the corresponding conversion (or convertibility) relations unchanged.

Much of the importance of *conversion* derives from its connexion with *equational reasoning*. Informally speaking, by equational reasoning we mean the way equations are used and manipulated in mathematics, and, as a matter of fact, already in simple school algebra. The vital acts of inference are always making substitutions in equations that have been earlier established, and using such equations within arbitrary contexts. Now this is exactly what happens in a conversion.

Therefore we will find

$$s =_E t \quad \Leftrightarrow \quad E \vdash s = t \qquad (*)$$

Here $E \vdash s = t$ signifies that the equation $s = t$ can be derived by using the equations in E in a system of *equational logic*. Several formalizations of equational logic exist, all giving rise to the (same) relation of deductive consequence denoted here by $\vdash$. Equational logic will be studied in Chapter 7, where the equivalence $(*)$ will be established in Exercise 7.1.3. This property of the convertibility relation, that it captures derivability in equational logic, is sometimes called *logicality* (after Bertling and Ganzinger [1989]).

2.4.3. EXERCISE. Let $\mathcal{R} = (\Sigma, R)$ be a TRS.

(i) Prove that $t \twoheadrightarrow s$ in $\mathcal{R}^{\leftrightarrow}$ if and only if $t =_R s$.

(ii) Prove that $t =_R s \Leftrightarrow t =_{R^{\leftrightarrow}} s$.

(iii) Show that a conversion in $\mathcal{R}^{=}$, i.e. a reduction sequence in $\mathcal{R}^{\leftrightarrow}$, is also a conversion sequence of the original TRS $\mathcal{R}$ (according to Definition 1.1.2).

2.4.4. EXERCISE. Let E be a set of equations. The subset relation $\sqsubseteq$ modulo E is defined as follows: $s \sqsubseteq t$ if and only if there exists a context C such that $t =_E C[s]$. Prove that $\sqsubseteq$ is a quasi-order; show that (the strict part of) $\sqsubseteq$ is in general not well-founded.

2.5. Semantics

So far, terms have been considered as purely formal objects, and rewriting as a purely formal game that can be played with terms. Likewise with equations. Of course one mostly has an interpretation in mind. Knowing this interpretation can make it easier to get an intuitive grasp of the formalism. Moreover, it will sometimes be profitable to be able to use the meaning of terms in reasoning about a TRS. On the other hand, in many cases there is not one fixed meaning of the terms. There may be reasonable alternatives to an intended meaning. And sometimes it will even be hard to find out what an intended meaning would be at all. A case in point is CL.

The notion of a *model* for an equational system $\langle \Sigma, E \rangle$ we use is just what is also called a model in first-order predicate logic. Adhering to the tradition of universal algebra and equational specifications we speak of a model also as a Σ-*algebra* satisfying the set of equations E.

In a model a domain is fixed, and, moreover, interpretations of the n-ary function symbols as n-ary functions over that domain, and in particular of the constants as elements of the domain. Since the formal syntax of terms is derived from ordinary mathematical function notation, the meaning of a term will often be obvious. For example, nobody will be surprised when the interpretation of the term $S(S(A(M(S(S(0)),0),S(S(0))))))$ evaluates to the natural number 4, being the outcome of $((2 \times 0 + 2) + 1) + 1$.

2.5.1. Definition. (i) Let Σ be a signature. A Σ-*algebra* $\mathcal{A}$ is a non-empty set A, called the *domain*, together with functions $F^{\mathcal{A}} : A^n \to A$ for every n-ary function symbol $F \in \Sigma$. (If F is nullary, i.e., F is a constant, then $F^{\mathcal{A}} \in A$.)

The link between the function symbol F and its interpreation $F^{\mathcal{A}}$ can be made explicit by way of the *interpretation function* $I(F) = F^{\mathcal{A}}$. Then the Σ-algebra can be denoted by the triple $\mathcal{A} = \langle \Sigma, A, I \rangle$.

(ii) An *assignment* for a Σ-algebra $\mathcal{A}$ is a function $\theta : \mathit{Var} \to A$ from the set of variables to the domain of $\mathcal{A}$.

(iii) Given a Σ-algebra $\mathcal{A}$ and an assignment θ, a term $t \in \mathit{Ter}(\Sigma)$ is assigned a meaning $[\![t]\!]^{\mathcal{A},\theta} \in A$ by interpreting function symbols in t via the interpretation function I and variables via θ. Explicitly, we have the following inductive definition:

$$\begin{aligned} [\![x]\!]^{\mathcal{A},\theta} &= \theta(x) \\ [\![F(t_1,\ldots,t_n)]\!]^{\mathcal{A},\theta} &= F^{\mathcal{A}}(t_1^{\mathcal{A},\theta},\ldots,t_n^{\mathcal{A},\theta}) \end{aligned}$$

(iv) Variables in an equation $s = t$ are (implicitly) universally quantified. So we say that the equation $s = t$ is *valid* in $\mathcal{A}$, or, equivalently, $\mathcal{A}$ *satisfies* $s = t$, if for every assignment θ we have $[\![s]\!]^{\mathcal{A},\theta} = [\![t]\!]^{\mathcal{A},\theta}$. Notation: $\mathcal{A} \models s = t$.

(v) If every equation in a specification E is satisfied by $\mathcal{A}$, then $\mathcal{A}$ is called a *model* of E. We also say that $\mathcal{A}$ *satisfies* E. Notation: $\mathcal{A} \models E$.

(vi) A model of the equational specification $\mathcal{R}^=$ corresponding to a TRS $\mathcal{R}$ is also called a *model* of $\mathcal{R}$.

2.5.2. EXAMPLE. The 'natural' interpretation of the specification $\mathcal{N}_=^{AM}$ given in Table 2.5 is the model $\mathcal{N}$, with domain $\mathbb{N}$ and with interpretatios $0^{\mathcal{N}} = 0$, $S^{\mathcal{N}}(n) = n+1$, $A^{\mathcal{N}}(n,m) = n+m$, $M^{\mathcal{N}}(n,m) = n \cdot m$. This model $\mathcal{N}$ is sometimes called the *standard model* of $\mathcal{N}_=^{AM}$.

It is a crucial property of rewriting that it respects (is *sound* with respect to) meaning. That is, rewriting affects the syntactic form of a term only, not its meaning. Like in arithmetic, where for example the arithmetical expressions $8 + (6 - 3)$ and $8 + 3$ are just different syntactic representations of the same number, most efficiently represented as 11. This will follow from the soundness theorem for equational logic, to be proved in Chapter 7.

2.5.3. THEOREM (soundness of rewriting). (i) *Let $\mathcal{A} \models E$. Then we have the implication $t =_E s \;\Rightarrow\; \mathcal{A} \models t = s$.*

(ii) *Let $\mathcal{A} \models R^=$. Then we have the implication $t =_R s \;\Rightarrow\; \mathcal{A} \models t = s$.*

PROOF. For the proof of (i) we refer to Proposition 7.1.15 and (ii) follows immediately from (i). □

2.5.4. REMARK. One possible motivation for turning an equational theory $\mathcal{E}$ into a rewriting system is to get a better grasp of its models. If a *complete* term rewriting system $\mathcal{R}$ corresponding to $\mathcal{E}$ can be found, then the (closed) normal forms of $\mathcal{R}$ constitute an *initial* model of $\mathcal{E}$. See Section 7.2 for a detailed study of initial models.

Sometimes semantical considerations can be helpful in proving syntactic properties of TRSs. The following (easy) result can be used in a proof of UN or CR.

2.5.5. PROPOSITION. (i) *Let $\mathcal{A}$ be a model of the TRS $\mathcal{R}$ such that for all normal forms t, t' of $\mathcal{R}$*

$$\mathcal{A} \models t = t' \;\Rightarrow\; t \equiv t'$$

Then $\mathcal{R}$ has the property UN (uniqueness of normal forms).

(ii) *If the displayed implication holds only for ground normal forms t, t', then $\mathcal{R}$ will have the property ground-UN.*

PROOF. Trivial. □

The use of this proposition is illustrated in the following exercises. More semantic confluence tests can be found in Plaisted [1985], in the setting of TRSs modulo an equivalence relation.

2.5.6. Exercise. Consider the TRS $\mathcal{N}^{AM}$ from Example 2.3.1.

(i) Which are the closed normal forms?

(ii) Show that $\mathcal{N}^{AM}$ has the property ground-UN. (Hint: Make use of the natural semantic interpretation of $\mathcal{N}^{AM}$.)

(iii) Show that $\mathcal{N}^{AM}$ is ground-WN. (Hint: Show by induction on terms that each closed term can be reduced to one of the normal forms found in (i). It may be easier to start out by first restricting $\mathcal{N}^{AM}$ to the signature with only $0, S, A$, and with only the rules ρ_1, ρ_2.)

(iv) Conclude that $\mathcal{N}^{AM}$ is ground-CR.

See also Exercise 3.3.2.

2.5.7. Exercise. Show that the TRS

$$\{A(x,0) \to x, A(x,S(y)) \to S(A(x,y)), A(A(x,y),z) \to A(x,A(y,z))\}$$

is ground-CR but not CR.

2.6. Complete term rewriting systems and the word problem

The *word problem* for an equational specification (Σ, E) is to decide for arbitrary closed Σ-terms t, s whether or not $s =_E t$. The same question, but without the restriction to closed terms, is called the *uniform word problem*.

Word problems are an important example of an area where rewriting techniques can be applied. The crucial notion here is that of a *complete* TRS, i.e. a TRS which is SN and CR. The reason for this can be explained as follows.

Suppose for the equational specification (Σ, E) we can find a complete TRS (Σ, R) such that for all terms $s, t \in Ter(\Sigma)$

$$t =_R s \;\Leftrightarrow\; t =_E s \qquad (*)$$

In this case (Σ, R) will be called a TRS *for* (Σ, E). (See Definition 7.1.19.) Then (if R has finitely many rewrite rules only) we have a positive solution of the uniform word problem. The decision algorithm is simple. In order to decide whether $s =_E t$, reduce s and t to their respective normal forms s', t'. Then $s =_R t$ holds if and only if $s' \equiv t'$. On the basis of the above, it will be clear that an important question is how to find a complete TRS $\mathcal{R}$ for a given set of equations E such that $(*)$ holds.

2.6.1. Example. Consider the equational specification $\mathcal{B}$ of the Booleans given in Table 2.6. It seems natural to orient the equations from left to right. However, the TRS that results, although it is SN, is not CR. The diverging reduction steps

$$\neg(\mathit{false}) \;\leftarrow\; \neg(\neg(\mathit{true})) \;\rightarrow\; \mathit{true}$$

do not converge, since $\neg(\mathit{false})$ and true are normal forms. Nevertheless, a complete TRS for $\mathcal{B}$ can be obtained, by adding the extra reduction rule $\neg(\mathit{false}) \to \mathit{true}$ (see Exercise 2.7.18).

$e_1:$	$\neg(true)$	$=$	$false$
$e_2:$	$\neg(\neg(x))$	$=$	x
$e_3:$	$and(true, x)$	$=$	x
$e_4:$	$and(false, x)$	$=$	$false$
$e_5:$	$or(x, y)$	$=$	$\neg(and(\neg(x), \neg(y)))$

Table 2.6: Equational specification of the Booleans

This example and similar ones will be further analysed in Section 2.7. The pair of terms $\neg(false)$ and $true$ will be called there a *critical pair*. The addition of the rule $\neg(false) \to true$ is an extremely simple example of the technique of *critical pair completion*. This is a sophisticated method for constructing a complete TRS for (and from) a given equational specification, by the adoption of strategically chosen new reduction rules. Chapter 7 will be completely devoted to this method.

It should be pointed out, however, that no method for finding complete term rewriting systems from an arbitrary equational specification can be universally successful. This follows already from the fact that not every equational specification $\mathcal{E}$ (even if finite) has a decidable word problem. A famous example of a specification with undecidable word problem is the set of equations obtained from CL, combinatory logic, to be defined in Section 3.2. (Details on how to prove undecidability results for CL can be found in Section 5.4.) Other classic instances of unsolvable word problems arise from semi-groups. In Table 5.1 in Section 5.2 one finds a specification of a semigroup with undecidable word problem.

More surprising is the observation that there are equational specifications (Σ, E) *with* decidable uniform word problem but *without* a complete TRS (Σ, R) satisfying $(*)$. An example is given in the following exercise.

2.6.2. Exercise. Let (Σ, E) be the specification given by the equations

$$\begin{aligned} Times(x, 0) &= 0 \\ Times(x, y) &= Times(y, x) \end{aligned}$$

(i) For $t \in Ter(\Sigma)$, we define the equivalence class $E^{\sim}(t) \subseteq Ter(\Sigma)$. There is one equivalence class with all terms containing the constant 0: if $0 \subseteq t$ then $E^{\sim}(t) = \{s \in Ter(\Sigma) \mid 0 \subseteq s\}$. For the terms without an occurrence of 0 we have the inductive definition

(1) $E^{\sim}(x) = \{x\}$ for every variable x,
(2) $E^{\sim}(Times(t, t')) = \{Times(s, s') \mid s \in E^{\sim}(t), s' \in E^{\sim}(t')\} \cup \{Times(s', s) \mid s \in E^{\sim}(t), s' \in E^{\sim}(t')\}$.

Show that $s =_E t$ if and only if $s \in E^{\sim}(t)$ (or equivalently $E^{\sim}(s) = E^{\sim}(t)$), and conclude that the uniform word problem problem for (Σ, E) is decidable.

(ii) Prove that there is no complete TRS (Σ, R) for (Σ, E), i.e. such that for all terms $s, t \in Ter(\Sigma)$, $s =_R t \Leftrightarrow s =_E t$. (Hint: Consider in a supposed complete

TRS the open terms $Times(x, y)$ and $Times(y, x)$ and their interpretations in the standard model.)

(iii) Find a TRS (Σ, R) for (Σ, E) that is ground-complete.

According to the last part of this exercise, it is in this case crucial for the incompletability that we have also considered equations $s = t$ between possibly *open* Σ-terms (i.e. containing variables).

There are also finite equational specifications (Σ, E) which do have decidable word problem (for ground terms), but for which no ground-complete TRS $\mathcal{R}$ exists. The simplest example is the specification $\mathcal{E}$ consisting of a single binary commutative operator $+$ and a constant 0, and a single equation $x + y = y + x$. For the same reason as in Exercise 2.6.2 no complete TRS $\mathcal{R}$ can be found for $\mathcal{E}$. But now we also have the stronger result that no finite TRS $\mathcal{R}$ exists which is ground-complete and such that for ground terms s, t we have $s =_R t \Leftrightarrow s =_E t$. See Exercise 7.1.20 in the chapter on the completion of equational specifications, where this matter will be further elaborated.

2.7. Overlap and critical pairs

One term can contain several occurrences of redexes, thereby offering a choice of which redex to contract first. This may lead to non-confluence, as we have seen e.g. in Example 2.6.1, but not necessarily so. So the natural question is: when is the interaction of redexes harmless, and when harmful? In this section we will address that question. The observations that we make here are fundamental for the study of confluence and completeness, but in fact are only a starting point. Several of the later chapters, in the first place Chapters 4 (on orthogonality) and 7 (on completion), find their origin in the issues raised here.

The basic notions of this section are that of *overlap*, between redexes or between reduction rules, and that of *critical pair*. Critical pairs arise when there is interference of redex occurrences within a given term. Such interference is called overlap.

2.7.1. Overlapping redex patterns

We will first give two simple examples explaining overlap.

2.7.1. Example. Consider the TRS with the following two reduction rules:

$$\begin{array}{lrcl} \rho_1: & F(G(x), y) & \to & x \\ \rho_2: & G(a) & \to & b \end{array}$$

This TRS is not confluent. The term $F(G(a), x)$ can be reduced in two ways. We have $F(G(a), x) \to_{\rho_1} a$ and also $F(G(a), x) \to_{\rho_2} F(b, x)$, where the terms a and $F(b, x)$ are clearly normal forms, and hence without a common reduct.

If we take a closer look at this example, we see that we have two redexes, $F(G(a), x)$ and $G(a)$, one containing the other so tightly that the redexes share a function symbol, the symbol G, that is part of the left-hand sides of both reduction rules involved. For both redexes one could say that G is essential, in the sense that without G it would not be a redex. Now, contraction of one of the two redexes destroys the other one by taking away that essential function symbol. This phenomenon will be called *overlap*. Before giving a precise definition we use another example to illustrate that one redex can also contain another *without* overlap.

2.7.2. EXAMPLE. Consider the TRS with reduction rules

$$\begin{array}{lllll} \rho_1: & F(x,y) & \to & x \\ \rho_2: & G(a) & \to & b \end{array}$$

Now the left-hand sides of the rules have no function symbol in common. So it is clearly impossible that two redexes share an essential function symbol in the way of the previous example. Of course we can still have a redex containing another, as in the term $F(G(a), x)$ again. We have $F(G(a), x) \to_{\rho_1} G(a)$ and $F(G(a), x) \to_{\rho_2} F(b, x)$. However, the results $G(a)$ and $F(b, x)$ can now both be further reduced to the common reduct b.[2] This harmless form of one redex containing another, without overlap, will be called *nesting*.

Examples 2.7.1 and 2.7.2 describe the two typical cases of a redex containing another redex. Either one or more essential symbols are shared, in which case we say that there is *overlap*, or not, and then the redexes are *nested*. We still have to make precise what is meant by sharing an essential function symbol. This can be done by defining the *pattern* of a redex as the fixed part of the left-hand side l of the corresponding reduction rule. That is, the prefix of l that results if its variables are left out. In Example 2.7.1 the patterns are $F(G(\Box), \Box)$ and $G(a)$. Patterns and overlap are now formally defined.

2.7.3. DEFINITION (pattern and arguments). (i) The *pattern* of a reduction rule $\rho : l \to r$ is defined as the context l^ϵ, where ϵ is the substitution defined by $\epsilon(x) = \Box$ for all variables x, i.e. replacing variables by holes.

(ii) Let s be a redex according to the reduction rule $\rho : l \to r$, and let P be the pattern of ρ. Then P is also the pattern of the redex s, a *redex pattern*. We have $l \equiv P[x_1, \ldots, x_n]$ and $s \equiv l^\sigma \equiv P[x_1^\sigma, \ldots, x_n^\sigma]$, for certain variables $x_1, \ldots, x_n$ and substitution σ. The terms $x_1^\sigma, \ldots, x_n^\sigma$ are called the *arguments* of redex s.

[2] As a matter of fact the resulting terms $G(a)$ and $F(b, x)$ are themselves redexes, which can both be seen as leftovers of a redex in the original term $F(G(a), x)$. The redex $G(a)$ is even an exact copy. This is not the case for $F(b, x)$, but here the head symbol F is obviously a copy of the head symbol of the original $F(G(a), x)$.

Leftovers of redexes in this sense will be called *residuals* later; the detailed analysis of residuals plays a key role in the study of *orthogonality*, see Chapter 4.

(iii) The terminology of pattern and arguments is also used when dealing with redex occurrences. Note that the pattern of a redex occurrence in t consists of a connected set of symbol occurrences in the term tree of t (e.g. the shaded area in Figure 2.3).

(iv) Two redex occurrences in a term t *overlap* if their patterns share at least one symbol occurrence. Here we do not count the *trivial overlap* between a redex s and itself, unless s is a redex with respect to two different reduction rules.

(v) The notion of overlap extends to reduction rules allowing redex overlap. Two reduction rules ρ_1, ρ_2 *overlap* if in a term t there are occurrences of a ρ_1-redex s_1 and a ρ_2-redex s_2, such that s_1 and s_2 overlap.

According to this definition the patterns of the rules of Example 2.7.1 indeed are $F(G(\Box), \Box)$ and $G(a)$. The pattern of rule ρ_1 of Example 2.7.2 is $F(\Box, \Box)$. In a term tree a pattern can for example be represented by grey-scaling. This is done for the pattern $F(G(\Box, S(0)), \Box, H(\Box))$ of the reduction rule $F(G(x, S(0)), y, H(z)) \to x$ in Figure 2.3.

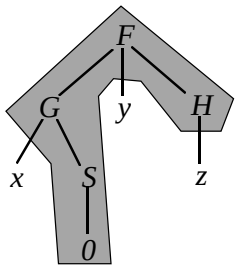


Figure 2.3: Example of a pattern

2.7.4. Example. Consider the TRS with reduction rules

$$\begin{array}{lll} \rho_1: & F(G(x, S(0)), y, H(z)) & \to \quad x \\ \rho_2: & G(H(x), S(y)) & \to \quad y \end{array}$$

In the term $F(G(H(0), S(0)), y, H(G(x, x)))$ we have overlapping redexes with patterns $F(G(\Box, S(0)), \Box, H(\Box))$ and $G(H(\Box), S(\Box))$, as depicted in Figure 2.4.

We continue with some more examples of overlapping redexes, which are illustrated in Figure 2.5. In Examples 2.7.5(ii) and (iii) the root (head) symbol of the overlapping redexes is shared; this is called *root overlap*.

2.7.5. Examples (Overlapping redex patterns). (i) Overlap is possible already if there is only one reduction rule. In the TRS $\{L(L(x)) \to 0\}$ consider the term $L(L(L(x)))$.

(ii) An example of overlap at the root, arising from the rules $F(0, x, y) \to 0$, $F(x, 1, y) \to 1$, is in the term $F(0, 1, x)$.

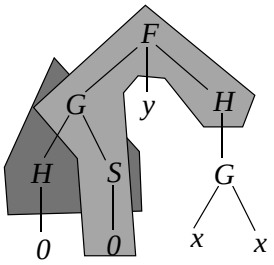


Figure 2.4: Example of overlap

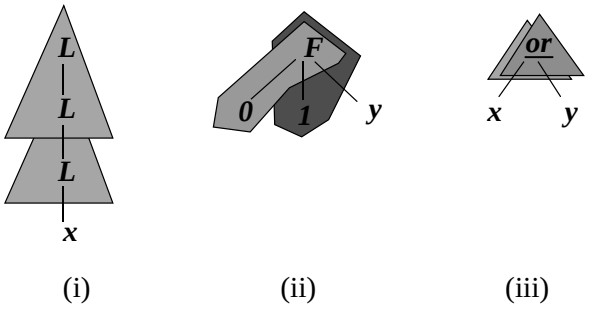


Figure 2.5: Overlapping redex patterns

(iii) Another example of root overlap is the redex $or(x, y)$ in the TRS with rules for non-deterministic choice, $\{or(x, y) \to x,\ or(x, y) \to y\}$.

We take a closer look at the phenomenon of overlap, from the perspective of the geometry of terms. In Subsection 2.1.1 we already observed quite in general that if two occurrences within a term t share a symbol occurrence then one of the two contains the other. In particular we will have this situation in the case of overlapping redexes, sharing a symbol of their patterns. Conversely, as we saw in Examples 2.7.1 and 2.7.2 above, if a redex contains another one, this can be either with or without overlap. These two possibilities are further analysed in the following lemma. It shows that we have overlap if and only if the inner redex happens to be an instance of a *non-variable* subterm of the left-hand side of the reduction rule corresponding to the outer redex.

2.7.6. Lemma. *Let s_0, s_1, such that $s_1 \leq s_0$, be redexes, according to reduction rules $\rho_0 : l_0 \to r_0$ with substitution σ, and $\rho_1 : l_1 \to r_1$ with substitution τ, respectively. So we have $s_0 \equiv {l_0}^\sigma$ and $s_1 \equiv {l_1}^\tau$. Then for the position of s_1 in s_0 there are the following two mutually exclusive possibilities.*

(i) *For some contexts D, E we have $l_0 \equiv D[x]$ (so $s_0 \equiv D^\sigma[x^\sigma]$) and s_1 occurs with prefix $D^\sigma[E[\,]]$ in s_0, that is $x^\sigma \equiv E[s_1]$.*

(ii) *There are a term s, not a variable, and a context $D[\,]$ such that $l_0 \equiv D[s]$ (so $s_0 \equiv D^\sigma[s^\sigma]$) and s_1 occurs with prefix $D^\sigma[\,]$ in s_0, that is $s_1 \equiv s^\sigma$.*

PROOF. Let P be the pattern of ρ_0; then l_0 is of the form $P[x_1, \ldots, x_n]$, the context P containing no further variables. We have $s_0 \equiv P[x_1{}^\sigma, \ldots, x_n{}^\sigma]$. There are two possibilities.

(i) Perhaps s_1 is contained in one of the $x_i{}^\sigma$'s, that is, s_1 occurs with a prefix $P[x_1{}^\sigma, \ldots, E[\ ], \ldots, x_n{}^\sigma]$. We are in case (i) with $D[\ ] \equiv P[x_1, \ldots, [\ ], \ldots, x_n]$.

(ii) Otherwise the head symbol of s_1 must occur in P, say at position p. We are in case (ii), with $s \equiv l_0|_p$ and $D[\ ] \equiv l_0[\ ]_p$. □

In case (i) of Lemma 2.7.6 we say that the redexes are *nested*; and that s_1 occurs *inside the substitution* in the redex s_0, or in one of its *arguments*. In case (ii) the redexes *overlap* according to Definition 2.7.3(iv), and according to 2.7.3(v) the corresponding reduction rules are also said to overlap. This gives us the following characterization of rule overlap.

2.7.7. LEMMA. *Two reduction rules $\rho_0 : l_0 \to r_0$ and $\rho_1 : l_1 \to r_1$ overlap if and only if there is a non-variable subterm of l_0 that can be matched with a ρ_1-redex (or, vice versa, a non-variable subterm of l_1 that can be matched with a ρ_0-redex). So we have $l_0 \equiv D[s]$ and $s^\sigma \equiv {l_1}^\tau$ for some context $D[\ ]$, term s, not a variable, and substitutions σ, τ. Here one does not count the* trivial *overlap between a rule $\rho : l \to r$ and itself on account of the trivial match between l and some ρ-redex l^τ.*

PROOF. Immediate by Lemma 2.7.6. □

Note that the case that ρ_0 and ρ_1 are the same rule is not excluded, but then $D[\ ]$ should not be the empty context.

2.7.2. Critical pairs and the Critical Pair Lemma

Overlap may lead to non-confluence, as was illustrated by Example 2.7.1. But not necessarily so. Here is a simple example of a TRS with overlap, which nevertheless can be shown to be confluent.

2.7.8. EXAMPLE. Consider the TRS with the following two reduction rules:

$$\begin{array}{lrcl} \rho_1: & F(G(x)) & \to & x \\ \rho_2: & G(x) & \to & G(x) \end{array}$$

We have overlapping redexes in $F(G(0))$, giving rise to 'diverging' reduction steps $F(G(0)) \to_{\rho_1} 0$ and $F(G(0)) \to_{\rho_2} F(G(0))$. Another application of $F(G(0)) \to_{\rho_1} 0$ makes the diverging steps converge again, however. It is not difficult to see this TRS is CR.

The examples we have seen so far suggest that if we want to investigate whether a TRS is confluent, it is sensible to start by looking at what happens when overlapping redexes are contracted. At first sight this may seem a hopeless task. In general there is the possibility of infinitely many substitutions,

and accordingly there will then be infinitely many cases of redex overlap to consider. It is not that bad, however. With each pair of overlapping reduction rules one can associate a typical, most general, case of redex overlap, from which other cases can be obtained by substitution. Contracting the overlapping redexes in this most general case results in a so-called *critical pair*. In this and the next subsection we will see that a lot can be learned of a TRS by just inspecting its critical pairs.

In the literature critical pairs are mostly defined using most general unifiers. We prefer a definition via the notion of most general common instance, since it is more direct. It remains a rather technical business, however. What should stick is that a critical pair consists of terms that result by contracting overlapping redexes, say with respect to rewrite rules ρ_1, ρ_2. Moreover, all other results of contracting overlapping ρ_1- and ρ_2-redexes can be obtained as substitution instances of the critical pair.

The critical pair corresponding to the overlapping rules ρ_1, ρ_2 in Example 2.7.8 would be $\langle F(G(x)), x\rangle$. The terms $F(G(0))$ and 0 that were the endpoints of the diverging reduction steps in that example can be obtained from the critical pair by substituting 0 for x.

2.7.9. Definition (critical pair). Consider a pair of overlapping reduction rules $\rho_0 : l_0 \to r_0$ and $\rho_1 : l_1 \to r_1$. By Lemma 2.7.7 (assuming ρ_0, ρ_1 are given in the appropriate order) there are a context $D[\ ]$, a term s, not a variable, and substitutions σ, τ such that $l_0 \equiv D[s]$ and $s^\sigma \equiv {l_1}^\tau$.

We may assume the substitutions σ, τ to be such that $s^\sigma \equiv {l_1}^\tau$ is an mgci of s, l_1. Without loss of generality, we require that σ is minimal in the sense that $Dom(\sigma) = Var(s)$, and that the variables introduced by σ are new in the sense that $Var(s^\sigma) \cap Var(D) = \varnothing$.

The pair of one-step reducts of the outer redex ${l_0}^\sigma \equiv D^\sigma[{l_1}^\tau]$ that arises from this overlap, $\langle D^\sigma[{r_1}^\tau], {r_0}^\sigma\rangle$, is called a *critical pair*.

Obviously, critical pairs are unique up to renaming. The order of the terms in a critical pair may be significant in some applications. So then the definition should be taken literally in this respect. Mostly not, however, and then some sloppiness may be permitted. There does not appear to be a standard order throughout the literature.[3] Note, by the way, that in a case of *root* overlap our definition gives rise to two critical pairs. For example, in the TRS with reduction rules $\{a \to a,\ a \to b\}$ one obtains both $\langle a, b\rangle$ and $\langle b, a\rangle$ as critical pairs.

2.7.10. Examples. (i) The rules of the TRS of Example 2.7.1 have the critical pair $\langle F(b, y), a\rangle$.

[3]The order of the elements of a critical pair in Definition 2.7.9 is that of Huet [1980] and Klop [1992]. This contrasts with Dershowitz and Jouannaud [1990] and Baader and Nipkow [1998], who use the reverse order.

(ii) The reduction rules $\neg(\mathit{true}) \to \mathit{false}$ and $\neg(\neg(x)) \to x$, which result from the first two equations in Example 2.6.1, give rise to the critical pair $\langle \neg(\mathit{false}), \mathit{true} \rangle$.

(iii) The TRS of Example 2.7.2 has no critical pairs.

2.7.11. EXERCISE. Consider the group equations $e \cdot x = x$, $I(x) \cdot x = e$, $(x \cdot y) \cdot z = x \cdot (y \cdot z)$, which can be turned into a TRS by orienting the equations from left to right:

$$\begin{array}{lll} \rho_1: & e \cdot x & \to & x \\ \rho_2: & I(x) \cdot x & \to & e \\ \rho_3: & (x \cdot y) \cdot z & \to & x \cdot (y \cdot z) \end{array}$$

Show that this TRS is not confluent and determine the critical pairs. (This example will be thoroughly analysed in Chapter 7.)

One now easily verifies that all pairs of overlapping redexes can be found as substitution instances of mgci's as in Definition 2.7.9. What we are interested in, then, are the instances of the critical pairs that arise this way.

2.7.12. LEMMA. *Let s_0, s_1 be overlapping redexes, $s_1 \leq s_0$. Then this redex overlap arises from an overlap between the corresponding reduction rules $\rho_0 : l_0 \to r_0$, $\rho_1 : l_1 \to r_1$, say with critical pair $\langle c_1, c_0 \rangle$.*

Assume, moreover, that $t_1 \overset{s_1}{\leftarrow} s_0 \overset{s_0}{\to} t_0$. Then the reducts t_0, t_1 can be found as instances of the critical pair, $t_0 \equiv {c_0}^{\rho}, t_1 \equiv {c_1}^{\rho}$, with the same substitution ρ.

PROOF. We have a context $D[\,]$, a term s, not a variable, and substitutions σ, τ, such that $s_0 \equiv {l_0}^{\sigma}$, $l_0 \equiv D[s]$ and $s^{\sigma} \equiv {l_1}^{\tau} \equiv s_1$. Let σ', τ' be substitutions such that $s^{\sigma'} \equiv l_1^{\tau'}$ is an mgci of s, l_1, and satisfying the requirements $\mathit{Dom}(\sigma') = \mathit{Var}(s)$ and $\mathit{Var}(s^{\sigma'}) \cap \mathit{Var}(D) = \varnothing$. So there is a substitution ρ such that $s^{\sigma'\rho} \equiv s^{\sigma} \equiv {l_1}^{\tau} \equiv {l_1}^{\tau'\rho}$. The substitution ρ can be chosen such that $D^{\sigma'\rho} = D^{\sigma}$: on variables that occur in s the substitutions $\sigma'\rho$ and σ already coincide; for other variables in D, untouched by σ', the choice of ρ is free.

The critical pair that arises from this overlap is $\langle D^{\sigma'}[r_1^{\tau'}], r_0^{\sigma'} \rangle$. We have $t_0 \equiv {r_0}^{\sigma} \equiv {r_0}^{\sigma'\rho}$ and $t_1 \equiv D^{\sigma}[{r_1}^{\tau}] \equiv D^{\sigma'\rho}[{r_1}^{\tau'\rho}] \equiv D^{\sigma'}[{r_1}^{\tau'}]^{\rho}$. □

2.7.13. EXERCISE. Determine the critical pairs that correspond to the cases of overlap in Example 2.7.5.

It is important to note that one pair of reduction rules can give rise to several cases of overlap, and thus to several, essentially different, critical pairs. It is instructive to construct an example of this (exercise).

At the beginning of this section we stated that critical pairs are important in analysing the confluence behaviour of a TRS. We observed (in Example 2.7.8) that overlap and, as a result of that, the presence of critical pairs does not necessarily result in non-confluence. However, it often does. The

following definition and lemma, the well-known Critical Pair Lemma, elaborate on this issue. As a matter of fact they establish a fundamental insight, which will serve as a basis for many of the later theoretical developments.

2.7.14. DEFINITION. A critical pair $\langle s, t\rangle$ is called *convergent* if s and t have a common reduct.

Obviously, if a non-convergent critical pair exists, then the TRS is not confluent. On the other hand, convergence of all critical pairs is not in general sufficient for confluence, but it will guarantee WCR. The latter is the content of the Critical Pair Lemma of Knuth and Bendix [1970] and Huet [1980].

2.7.15. LEMMA (Critical Pair Lemma). *A TRS $\mathcal{R}$ is* WCR *if and only if all its critical pairs are convergent.*

PROOF. One direction is obvious: the elements of a critical pair are one-step reducts of the outer one of two overlapping redexes. The common reduct exists by WCR.

For the other direction, given a peak $t_0 \overset{s_0}{\leftarrow} t \overset{s_1}{\rightarrow} t_1$, we have to find a common reduct t_2 of t_0 and t_1. Let s_0', s_1' be the respective contracta of s_0, s_1. Distinguish cases according to the relative positions of the redexes s_0 and s_1 in t. Each case is illustrated by one of the sketches in Figure 2.6.

(a) Suppose s_0 and s_1 are *disjoint* redex occurrences in t. Then there is a context C such that $t \equiv C[s_0, s_1]$. So $t_0 \equiv C[s_0', s_1]$ and $t_1 \equiv C[s_0, s_1']$. Take $t_2 \equiv C[s_0', s_1']$.

(b) Suppose s_0 and s_1 are *nested*. Without loss of generality we assume that s_1 is a subterm of s_0. Let $C[\,]$ be the prefix of s_0 in t and assume s_0 to be a redex according to the rewrite rule $l_0 \to r_0$. Then for some substitution σ we have $s_0 \equiv {l_0}^\sigma$ and $s_0' \equiv {r_0}^\sigma$. Since there is no overlap, the redex s_1 occurs inside one of the substitutions in ${l_0}^\sigma$. I.e., $l_0 \equiv D[x]$, $s_0 \equiv D^\sigma[x^\sigma]$ and $x^\sigma \equiv E[s_1]$. Then $t_1 \equiv C[D^\sigma[E[s_1']]]$. Define the substitution σ' by $\sigma'(x) \equiv E[s_1']$ and $\sigma'(y) \equiv \sigma(y)$ if $y \not\equiv x$. Now if l_0 is left-linear, that is, if the variable x does not occur in $D[\,]$, it is easy: $t_1 \equiv C[{l_0}^{\sigma'}] \to C[{r_0}^{\sigma'}] \twoheadleftarrow C[{r_0}^\sigma] \equiv t_0$. If l_0 is not left-linear, we may need some extra reduction steps to obtain $t_1 \twoheadrightarrow C[{l_0}^{\sigma'}] \to C[{r_0}^{\sigma'}]$; see Figure 2.6(b′). Anyhow, in both cases we can take $t_2 \equiv C[{r_0}^{\sigma'}]$.

(c) Suppose the redexes *overlap.* If the overlap is trivial, that is, if s_0 and s_1 coincide and are redexes according to the same reduction rule, then obviously $t_0 \equiv t_1$, and t_2 can be taken as that very term.

The interesting case is when the overlap is non-trivial. Again assume that s_1 is a subterm of s_0 and that $C[\,]$ is the prefix of s_0. By Lemma 2.7.12 the reducts t_0, t_1 can be found as instances ${c_0}^\rho, {c_1}^\rho$, for some critical pair $\langle c_1, c_0\rangle$ of $\mathcal{R}$. By assumption the critical pair is convergent, say with common reduct s_2. Take $t_2 \equiv C[{s_2}^\rho]$. □

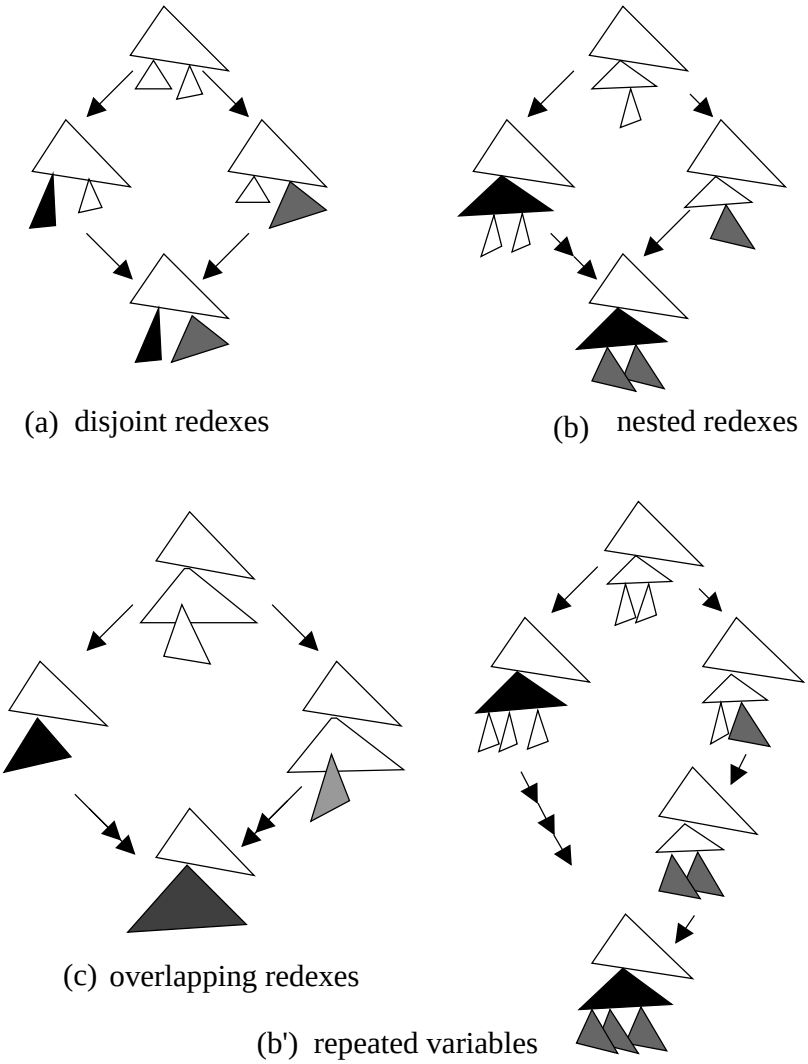


Figure 2.6: Critical Pair Lemma

The following theorem of Knuth and Bendix [1970] is an immediate corollary of the Critical Pair Lemma (2.7.15) and Newman's Lemma (1.2.1):

2.7.16. THEOREM. *Let $\mathcal{R}$ be a TRS which is* SN. *Then $\mathcal{R}$ is* CR *if and only if all critical pairs of $\mathcal{R}$ are convergent.*

PROOF. Exercise. □

2.7.17. EXERCISE. Consider a TRS with the following rewrite rules.

$$\begin{aligned} x \cdot \varnothing &\to x \\ \varnothing \cdot x &\to x \\ x/\varnothing &\to x \\ \varnothing/x &\to \varnothing \end{aligned}$$

$$\begin{array}{rcl} x/x & \to & \varnothing \\ (x \cdot y)/z & \to & (x/z) \cdot (y/(z/x)) \\ z/(x \cdot y) & \to & (z/x)/y \end{array}$$

It turns out that this TRS is terminating (SN). Prove that also CR holds, by considering the critical pairs. (This TRS axiomatizes projection equivalence of reductions; see Subsection 4.6.3.)

2.7.18. Exercise. Consider the following TRS, derived from the equational specification of the Booleans of Table 2.6.

$$\begin{array}{rlcl} r_1: & \neg(\mathit{true}) & \to & \mathit{false} \\ r_2: & \neg(\neg(x)) & \to & x \\ r_3: & \mathit{and}(\mathit{true}, x) & \to & x \\ r_4: & \mathit{and}(\mathit{false}, x) & \to & \mathit{false} \\ r_5: & \mathit{or}(x, y) & \to & \neg(\mathit{and}(\neg(x), \neg(y))) \\ r_6: & \neg(\mathit{false}) & \to & \mathit{true} \end{array}$$

Prove that this TRS is complete by analysing the critical pairs and proving SN.

2.7.19. Exercise. Prove, using the Critical Pair Lemma: if the TRS $\mathcal{R}$ has finitely many rules and is SN, then WCR and CR for $\mathcal{R}$ are decidable.

2.7.3. Roads to confluence

Now that it has become clear that critical pairs are a potential source of non-confluence, it is time to see an example where the absence of critical pairs does not guarantee confluence either. The following example is from Klop [1980].

2.7.20. Exercise. Consider the TRS $\mathcal{R}$, consisting of the reduction rules

$$\begin{array}{rlcl} \rho_1: & D(x,x) & \to & E \\ \rho_2: & C(x) & \to & D(x, C(x)) \\ \rho_3: & A & \to & C(A) \end{array}$$

(i) Verify that $\mathcal{R}$ does not have critical pairs.

(ii) Conclude that $\mathcal{R}$ is WCR.

(iii) Show that $C(A) \twoheadrightarrow E$ and $C(A) \twoheadrightarrow C(E)$.

(iv) Show that $\mathcal{R}$ is not CR. (Hint: Show that the terms E and $C(E)$ do not have a common reduct.)

Closer inspection of the TRS of Exercise 2.7.20 shows that it is the presence of the non-lef-tlinear rule ρ_1 that spoils CR in this particular case. So one might conjecture that term rewriting systems without critical pairs *and* with only left-linear reduction rules are confluent. This turns out to be the case indeed. Such systems are called *orthogonal*. Orthogonal TRSs form an important class of term rewriting systems, which will be studied in detail in Chapter 4.

So one strategy to ensure confluence when devising a TRS is: avoid critical pairs altogether (and also non-left-linear rules). The result will be an orthogonal system. There is also a second main strategy, which is based on the Knuth-Bendix Theorem. The aim is to construct a TRS that is SN and of which all critical pairs are convergent. Such TRSs are complete by Theorem 2.7.16. This method is called *completion* (or, more fully: *critical pair* or *Knuth–Bendix* completion). It will be studied extensively in Chapter 7. Here we give just a brief outline.

Starting out with an equational specification, turn the equations into rewrite rules. Make sure that this is done in such a way that the resulting TRS is terminating. (An example would be the oriented group axioms of Exercise 2.7.11.) Then, when there is a non-convergent critical pair, add rewrite rules that produce a common reduct for the terms in the critical pair. One should be careful that termination is preserved by the new rules. Repeat this process of adding rules as long as non-convergent critical pairs are encountered. Since the addition of an extra rewrite rule may give rise to new critical pairs, which also have to be taken care of, this procedure may take some time and cannot be guaranteed to terminate. However, in many practical cases the method is successful.

3

Examples of TRSs and special rewriting formats

Jan Willem Klop
Roel de Vrijer

In this chapter some more extended examples of term rewriting sytems are presented. Typical examples are TRSs containing numerals, i.e. representations of the natural numbers. We introduce the concept of stream, illustrating that it is not always only the normal form of a term that counts, but also its infinitary behaviour. The paradigm example of a TRS, combinatory logic, gets an extensive treatment in Section 3.2. As modifications of the first-order format string rewriting and many-sorted rewriting are briefly dicussed. The chapter concludes with a section about conditional rewriting.

3.1. Numerals and streams

A constant 0 and a unary function symbol S can encode the natural numbers as the terms $0, S(0), S(S(0)), \ldots$, as e.g. in the TRS $\mathcal{N}^{AM}$ in Example 2.3.1. These closed terms are then called *numerals*.[1]

The TRS $\mathcal{N}^{AM}$, with function symbols for addition and multiplication, can easily be extended to accommodate other arithmetic operations. For example, one could add a binary function symbol E for exponentiation, with the extra reduction rules $\{E(x,0) \to S(0),\ E(x,S(y)) \to M(E(x,y),x)\}$. Or a binary function symbol *Min* for minimum, with rules $\{Min(S(x),S(y)) \to S(Min(x,y)),\ Min(x,0) \to 0,\ Min(0,x) \to 0\}$. And so on.

In these examples a closed term is viewed as a terminating process that, when computed (reduced), yields a numeral as outcome. There is yet another possibility: a term can code an infinite process, e.g. generating an infinite number sequence (a *stream*). We make use of a binary function symbol *Cons*, written as : in infix notation, for adding an element at the front of a list. Now consider a unary function symbol *nats* with reduction rule

$$nats(x) \to x : nats(S(x))$$

Starting with $nats(0)$ we obtain the infinite reduction sequence

$$nats(0) \ \to\ 0 : nats(S(0)) \ \to\ 0 : S(0) : nats(S(S(0)) \ \to\ \cdots$$

[1] It is common practice to use the term 'numerals' for a formal representation of the natural numbers within a formal system.

Step by step this reduction generates the natural number sequence. So a term like $nats(0)$ can be seen as representing a stream. In the limit, this reduction converges to an infinite term

$$0 : S(0) : S(S(0)) : S(S(S(0))) : \cdots$$

In Chapter 12 of Terese [2003], which is devoted to infinitary rewriting, the notion of an infinite term as the limit of a converging infinite reduction is made precise.

We give another simple example of a stream. A constant *zeros* with reduction rule $zeros \to 0 : zeros$ generates the infinite sequence $0 : 0 : 0 : 0 : \cdots$. Also operations on streams can be represented. For example, a binary function symbol *add* with reduction rule $add(x_1 : y_1, x_2 : y_2) \to A(x_1, x_2) : add(y_1, y_2)$ expresses the operation of adding two streams of numerals elementwise.

3.1.1. Exercise. (i) Define a TRS extending $\mathcal{N}^{AM}$ which has a closed term representing as a stream the sequence of Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, \ldots$.

(ii) Define in a TRS the operator *Search*, which given a stream of natural numbers as input computes the place of the first occurrence of the number 0, if it exists.

(iii) Define in a TRS the operator *Insert*, which given a stream representing an increasing sequence of natural numbers and a natural number n inserts n at the place where it belongs.

In the next exercise we will construct a non-terminating term *ham*, which rewrites to any finite initial part of the sequence of Hamming numbers. (For an introduction to Hamming numbers see Dijkstra [1976].)

3.1.2. Exercise. The Hamming numbers are all natural numbers that can be written in the form $2^i \cdot 3^j \cdot 5^k$, for some $i, j, k \geq 0$. The aim of this exercise is to construct a TRS generating, as a stream, the sequence of Hamming numbers in ascending order. So this sequence starts with

$$1\ 2\ 3\ 4\ 5\ 6\ 8\ 9\ 10\ 12\ 15\ \ldots$$

One may proceed in three steps.

(i) Construct a TRS computing the maximum of two natural numbers.

(ii) The merge, $merge(\sigma, \tau)$, of two ascending sequences σ, τ of natural numbers is the sequence that contains the elements of both σ and τ, in ascending order (so without repetitions). E.g.

$$merge(\langle 2, 3, 5\rangle, \langle 1, 3, 8\rangle) = \langle 1, 2, 3, 5, 8\rangle$$

Construct a TRS computing the merge of two ascending sequences of natural numbers.

(iii) Construct a TRS with a constant *ham* which generates in ascending order the sequence of Hamming numbers.

3.1.3. EXERCISE. The celebrated Thue–Morse sequence is a stream of 0's and 1's arising as follows. Start with 1, and append in each generation step the 'negative' of the sequence obtained thus far. The result is the stream

$$M = 1001\,0110\,0110\,1001\,0110\,1001\,1001\,0110\ \ldots$$

Construct a TRS generating M.

3.2. Combinatory logic

One could say that *the* paradigmatic example of a TRS is the system of combinatory logic (CL). As a matter of fact, the roots of the very notion of term rewriting, and of much of its theory, lie in the system CL. Combinatory logic originated in the 1920s, when M. Schönfinkel [1924] started studying the combinators with the aim of analysing the concept of substitution and eliminating the use of bound variables from logic and mathematics. Combinatory logic was further developed by H.B. Curry [1930]; see also Curry and Feys [1958]. His aim was in the spirit of Schönfinkel's: the elimination of the bound variable. He established CL as an important model of computation. Along with the λ-calculus, it contributed to the analysis of the notion of a computable, *recursive*, function and stands at the basis of the discipline of functional programming. In particular, combinatory logic can be fruitfully employed in the implementation of functional programming languages. See e.g. Turner [1979].

We will discuss CL at some length, in the first place as an instructive example. What counts especially in this respect is that CL is a very rich TRS, with an interesting, elegant and non-trivial theory. CL and some of its extensions can be profitably used to illustrate TRS notions and proof techniques.

In CL, but also in several other important TRSs, and especially in the λ-calculus, there is a very special binary operator, called *application* (Ap). One should think here of function application: in the term $Ap(t, s)$ the term t is considered as a function, with s as its argument. Notice that by the rules of term formation of a TRS, any term can be placed arbitrarily in either the function position or the argument position of an application, or even both at the same time, as in the term $Ap(t, t)$. This is called *self-application*. There are no type restrictions. Systems with application are sometimes called *applicative* TRSs. For a definition see Subsection 3.2.5, where some attention will be paid to the issue of applicative TRSs in a more general setting.

The signature of CL has, apart from Ap, only the three constants S, K, I. The rewrite rules are given in Table 3.1.
Usually one uses the infix notation $(t \cdot s)$ instead of $Ap(t, s)$, in which case the rewrite rules of CL read as in Table 3.2. Moreover, as in ordinary algebra, the dot is mostly suppressed. As a further notational simplification, not all brackets need to be written. That is, outermost brackets can be omitted;

$Ap(Ap(Ap(S,x),y),z)$	$\to$	$Ap(Ap(x,z),Ap(y,z))$
$Ap(Ap(K,x),y)$	$\to$	x
$Ap(I,x)$	$\to$	x

Table 3.1: Combinatory logic, functional notation

$(((S \cdot x) \cdot y) \cdot z)$	$\to$	$((x \cdot z) \cdot (y \cdot z))$
$((K \cdot x) \cdot y)$	$\to$	x
$(I \cdot x)$	$\to$	x

Table 3.2: Combinatory logic, infix notation

and in addition, certain pairs of brackets can be dropped according to the convention of *association to the left*. This means that in re-inserting the missing brackets, one chooses in each step of the restoration process the leftmost possibility. In particular, $t_1t_2t_3$ becomes $(t_1t_2)t_3$ and not $t_1(t_2t_3)$. The sum of all these conventions is known as *applicative notation*. The three reduction rules of CL in applicative notation are displayed in Table 3.3.

$Sxyz$	$\to$	$xz(yz)$
Kxy	$\to$	x
Ix	$\to$	x

Table 3.3: Combinatory logic, applicative notation

3.2.1. Examples. The expression $xz(yz)$ restores to $(xz)(yz)$, not to $x(z(yz))$. Likewise Kxy restores to $(Kx)y$, not $K(xy)$. Of course not all bracket pairs can be dropped: $xzyz$ is when restored $((xz)y)z$, which is quite different from $xz(yz)$. Note that the term SIx does not contain a redex Ix.

3.2.2. Exercise. Consider the CL-term $t \equiv SI(Kx)Iy$.

(i) Insert all missing brackets in t.

(ii) Write t out in full prefix notation, using Ap.

(iii) Indicate all redexes in t.

(iv) Reduce t to normal form.

At this point it can only be mentioned that CL has the Church–Rosser property. In Chapter 4 we will see that this follows because CL is a so-called *orthogonal* TRS. (See also Section 2.7 for a definition of orthogonal.) CL is not a complete TRS, however, since it fails to be SN. The following example exhibits a term admitting infinite reduction sequences.

3.2.3. Example. We have the reduction cycle $SII(SII) \to I(SII)(I(SII)) \to SII(I(SII)) \to SII(SII)$. The term $SII(SII)$ has many more reductions, which constitute an interesting reduction graph, drawn in Figure 3.1.

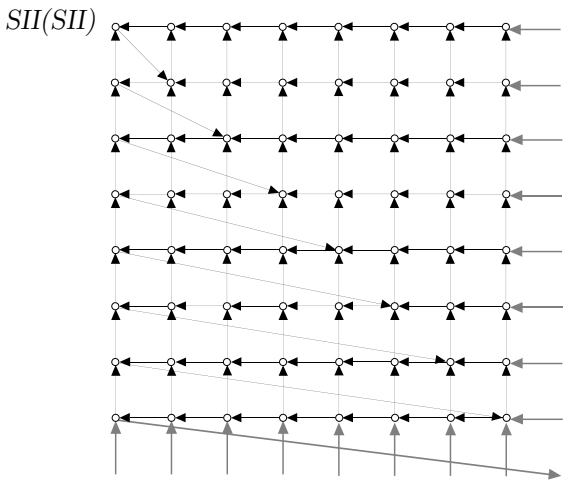


Figure 3.1: The reduction graph of $SII(SII)$

3.2.1. The combinators

In the applicative notation the constants S, K, I may be followed by an arbitrary number of arguments $t_1, \ldots, t_n$ $(n \geq 0)$. Therefore one may look at these constants as operators with variable arity. But note that, in the case of S, at least three arguments are needed before one can use the rewrite rule for S; for example, $St_1t_2t_3t_4t_5t_6 \to t_1t_3(t_2t_3)t_4t_5t_6$.

The constants I, K, S are called the *basic combinators*. Inspecting their behaviour as given in Table 3.3, we see that I acts as a 'universal' identity operator. Universal here means without domain restrictions. Application is defined between any pair of terms. Thus I can even be applied to itself, and we get $II \to I$. The basic combinator K may be thought of as a builder of constant functions. Applying K to an argument t yields the constant function Kt, which gives the same value t on any argument s. Equivalently K can be viewed as a projection function: given two arguments, it chooses the first one. The basic combinator S has a slightly more complicated behaviour; it is related to substitution. Its significance will become clear soon, when we prove combinatorial completeness.

3.2.4. Examples. Apart from I, K, S many more operators can be defined in CL. For example, composition of functions is represented by the term $B \equiv S(KS)K$. We have

$$Bxyz \equiv S(KS)Kxyz \quad \to \quad KSx(Kx)yz$$

$$\begin{aligned} &\to S(Kx)yz \\ &\to Kxz(yz) \\ &\to x(yz) \end{aligned}$$

Another example, the term $D \equiv SII$, was already encountered in Example 3.2.3. Given an arbitrary argument, D copies it and applies it to itself. We have

$$\begin{aligned} Dx \equiv SIIx &\to Ix(Ix) \\ &\to x(Ix) \\ &\to xx \end{aligned}$$

Likewise, defining C as $S(BBS)(KK)$, we have $Cxyz \twoheadrightarrow xzy$ as the reader may check.

Terms such as B, D, C, but in fact all closed CL-terms, are called *combinators*. The above examples of combinators are well-known examples, which were already introduced by Schönfinkel, and can be found in the books by Curry and Feys [1958], Barendregt [1984] or Hindley and Seldin [1986].[2]

Note that each of the above combinators is a solution for F in an explicit definition of the form

$$Fx_1 \dots x_n \twoheadrightarrow t,$$

with t a CL-term containing only variables from the set $\{x_1, \dots, x_n\}$. Not only these, but all such explicitly definable functions, can be represented within CL as combinators. This phenomenon is called *combinatorial completeness*, and forms the essence of CL. It was Schönfinkel's big discovery that combinatorial completeness could already be attained by the clever choice of a small set of basic combinators.

From the proof of Theorem 3.2.9 it will be apparent that the basic combinators I, K, S are just tailor-made for the purpose of combinatorial completeness. This proof gives the real motivation for their choice. As a matter of fact, one could be even more efficient and drop the basic combinator I, which can be defined in terms of S and K.

3.2.5. Exercise. Find a closed CL-term F with only the basic combinators S, K, such that $Fx \twoheadrightarrow x$.

Combinatorial completeness will be established by defining within CL so-called *abstraction* expressions $[x]t$, which behave like λ-abstractions $\lambda x.t$ in the λ-calculus. What that means is expressed in Proposition 3.2.10. For a more explicit treatment of the correspondence between CL and the λ-calculus see Chapter 10 of Terese [2003].

As a matter of fact, there are many ways to define $[x]t$. None of them are very efficient though: they tend to be exponential in the length of t.

[2] Constructing combinators and playing with them can be quite entertaining, or even "fun" (Scott [1975]). A nice collection of fantasies and puzzles featuring combinators is contained in Smullyan [1985].

3.2.6. DEFINITION. For each CL-term t and each variable x, the CL-term $[x]t$ is defined inductively by

(i) $[x]t \equiv Kt$, if t is a constant or a variable other than x,

(ii) $[x]x \equiv I$,

(iii) $[x]tt' \equiv S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$.

3.2.7. EXAMPLE. Let $t \equiv [y]yx$ and $t' \equiv [xy]yx$. Then

(i) $t \equiv S([y]y)([y]x) \equiv SI(Kx)$, and

(ii) $t' \equiv [x]t \equiv [x](SI(Kx)) \equiv S([x](SI))([x](Kx)) \equiv$
$S(K(SI))(S([x]K)([x]x)) \equiv S(K(SI))(S(KK)I)$.

3.2.8. PROPOSITION. (i) $([x]t)x \twoheadrightarrow t$.

(ii) *The variable x does not occur in the* CL*-term denoted by $[x]t$.*

PROOF. (i) We use induction on the structure of t. For the base case we have $([x]x)x \equiv Ix \to x$ and $([x]y)x \equiv Kyx \to y$; the same if t is a constant. For the induction step note that the induction hypothesis yields $([x]t)x \twoheadrightarrow t$ and $([x]t')x \twoheadrightarrow t'$, and using these reductions we obtain

$$([x]tt')x \equiv S([x]t)([x]t')x \to ([x]t)x(([x]t')x) \twoheadrightarrow t(([x]t')x) \twoheadrightarrow tt'$$

(ii) Again an easy induction on t. □

3.2.9. THEOREM (combinatorial completeness). *Given a* CL*-term t, one can find a* CL*-term F such that $Fx_1\ldots x_n \twoheadrightarrow t$.*

PROOF. Let $F \equiv [x_1x_2\ldots x_n]t$ and use induction on n. Proposition 3.2.8(i) and the induction hypothesis yield the reduction

$$Fx_1\ldots x_n \equiv ([x_1][x_2\ldots x_n]t)x_1\ldots x_n \twoheadrightarrow ([x_2\ldots x_n]t)x_2\ldots x_n \twoheadrightarrow t \qquad \square$$

3.2.10. PROPOSITION. *Let t, t' be* CL*-terms. Then $([x]t)t' \twoheadrightarrow t[x := t']$.*

PROOF. This follows immediately from Proposition 3.2.8 by substituting t' for x in the reduction found in (i), since, by (ii), the variable x does not occur in $[x]t$. One uses the substitution property, Exercise 2.3.5(ii). □

3.2.11. EXERCISE. Proposition 3.2.10 can be generalized to multiple arguments. Prove the following. Given a CL-term t, one can find a CL-term t' such that $t't_1\ldots t_n \twoheadrightarrow t[x_1 := t_1]\ldots[x_n := t_n]$ for all CL-terms $t_1, \ldots, t_n$ which do not contain any of the variables $x_1, \ldots, x_n$.

3.2.2. Miscellaneous facts about combinators

In this subsection we collect a few remarkable facts on CL. Some are well-known, others are more or less folklore. The presentation is in the form of exercises. We start with a definition.

3.2.12. Definition. (i) An *S-term* is a closed CL-term that has only occurrences of the constant S.

(ii) A CL-term is *flat* if it can be written without visible brackets by using the bracket convention of association to the left.

For example the term $SKSSSKS$ is flat, but the S-term $SSS(SSS)(SSS)$ is not. The flat S-terms are exactly the terms $S, SS, SSS, SSSS, \ldots$

The following exercise is about the flat S-terms. The result is also incorporated as a consequence of Exercise 3.2.14(iii), but in this form it is much simpler.[3]

3.2.13. Exercise. Consider the CL-terms S, SS, SSS, ...etc. Prove that all these terms are WN.

3.2.14. Exercise. (i) Show that the S-term $SSS(SSS)(SSS)$ has an infinite reduction.

(ii) Show that no S-term has a cyclic reduction, i.e. there is no reduction $M \to^+ M$ for any S-term M.

(iii) Prove that all flat closed CL-terms built from the constants S and K are strongly normalizing.

(iv) The restriction to S and K was essential in (iii). Give a counterexample to SN if I may also be used.

3.2.15. Exercise. Consider the infinite sequence of combinators

$$S \;\; SK \;\; SKS \;\; SKSK \;\; SKSKS \;\; SKSKSK \;\; \ldots$$

Some of these terms are convertible. How many equivalence classes (with respect to convertibility) are there?

3.2.16. Exercise (A. Visser). Prove that S-terms can 'eat' at most three of their arguments. More precisely, if s is an S-term, and x_i $(i = 1, \ldots, 4)$ are variables, then any CL-reduct of the term $sx_1x_2x_3x_4$ has the form $s'x_4$, where x_4 does not occur in s'.

A lot more on S-terms can be found in Waldmann [2000].

[3] Later, in Section 4.7, we will see that for non-erasing reduction WN and SN are equivalent (Theorem 4.7.5). Since the reduction rule for S is non-erasing, this exercise then in fact already establishes SN for flat S-terms.

3.2.3. Fixed points

Many combinators can be constructed, with sometimes remarkable behaviour. For example let F be an arbitrary CL-term. Consider $P_F \equiv D(BFD)$. We have

$$P_F \equiv D(BFD) \twoheadrightarrow BFD(BFD) \twoheadrightarrow F(D(BFD)) \equiv FP_F$$

By viewing this reduction as a conversion, we get $FP_F = P_F$; the term P_F behaves as a *fixed point* for F. (Cf. Definition A.1.2(vi).)

Note that the construction we gave of the fixed point is uniform in F in the sense that P_F can be seen as a CL-term containing F as a parameter. By combinatorial completeness, we can then find a combinator P such that $PF = P_F$ for any F, namely $P \equiv [x]D(BxD)$. Combinators yielding fixed points in this way are called *fixed-point combinators*.

3.2.17. Definition. A *fixed-point combinator* Y is any closed CL-term for which there is a conversion

$$Yx = x(Yx)$$

Actually there are many fixed-point combinators in CL. The most famous one is Curry's so-called *paradoxical combinator*

$$Y_C \equiv SSI(SB(KD))$$

3.2.18. Exercise. (i) Verify that for any CL-term F we have a conversion $F(Y_C F) = Y_C F$.

(ii) Show that the term $[f]D[x]f(xx)$ is a fixed-point combinator.

(iii) Prove that fixed-point combinators exist in any applicative TRS that is combinatorial complete.

(iv) Prove that for any fixed-point combinator Y, the CL-term $Y' \equiv Y(SI)$ is again a fixed-point combinator.

The importance of fixed-point combinators lies in the following. By combinatorial completeness we already have in CL the possibility of explicit function definition. Any defining equation for a function F has a solution. With the help of a fixed-point combinator this can be strengthened to *implicit* function definition. This means that in the defining equation for F, the symbol F itself may occur in the right-hand side. In particular, this yields the facility of giving recursive definitions within CL. The following theorem is a strengthening of Theorem 3.2.9. (It is instructive to compare the two theorems.)

3.2.19. Theorem (implicit function definition). *Given any* CL*-term* t*, one can find a* CL*-term* F *such that* $Fx_1 \dots x_n = t[y := F]$.

PROOF. Let $t' \equiv [y][x_1 \ldots x_n]t$ and define $F \equiv Yt'$ for some fixed-point combinator Y. Then we have $Fx_1 \ldots x_n = t'Fx_1 \ldots x_n \equiv t'yx_1 \ldots x_n[y := F] \twoheadrightarrow t[y := F]$. Here we used subsequently that $F = t'F$, that $y \notin Var(t')$, that $t'yx_1 \ldots x_n \twoheadrightarrow t$ and that reduction is closed under substitution. □

3.2.20. EXAMPLE. As an example of implicit function definition we construct a combinator Z with the property that it 'eats' its argument: $Zx = Z$. By applying Theorem 3.2.19 with $t \equiv y$ we see that such a Z exists. The construction in the proof yields $t' \equiv [y][x]y$ and $Z \equiv Y[y][x]y \equiv Y(S(KK)I)$. The reader may check that $Z \equiv YK$ would do too (since $S(KK)Ix = Kx$).

With the possibility of explicit and implicit function definition, and of implementing recursion, the TRS CL has 'universal computational power': every (partial) recursive function on the natural numbers can be expressed in CL. This feature, which is common to CL and the λ-calculus, is what gave these systems their historic importance. A consequence is that CL can be used to implement functional programming languages. This is done for example in Turner [1979]. Actually, an extension of CL is used there, called SKIM (for SKI-Machine). It is also an applicative TRS (in the sense of Subsection 3.2.5), extending CL with extra constants (see Table 3.4). The reader will recognize some of the constants, viz. B, C, Y, as new disguises of old combinators. In fact, the extra constants in SKIM are there for reasons of efficient implementation only; they can all be defined using only S and K.

$Sxyz$	$\to$	$xz(yz)$	
Kxy	$\to$	x	
Ix	$\to$	x	
$Cxyz$	$\to$	xzy	
$Bxyz$	$\to$	$x(yz)$	
Yx	$\to$	$x(Yx)$	
$P_0(Pxy)$	$\to$	x	
$P_1(Pxy)$	$\to$	y	
$Uz(Pxy)$	$\to$	zxy	
$Cond\ True\ x\ y$	$\to$	x	
$Cond\ False\ x\ y$	$\to$	y	
$Plus\ \mathbf{n}\ \mathbf{m}$	$\to$	$\mathbf{n+m}$	
$Times\ \mathbf{n}\ \mathbf{m}$	$\to$	$\mathbf{n \cdot m}$	
$Eq\ \mathbf{n}\ \mathbf{n}$	$\to$	$True$	
$Eq\ \mathbf{n}\ \mathbf{m}$	$\to$	$False$	if $n \neq m$

Table 3.4: SKIM

Note that the TRS SKIM has an infinite signature: there is a constant $\mathbf{n}$ for each $n \in \mathbb{N}$.[4] There are also infinitely many reduction rules, because the

[4]Note that the boldface expression $\mathbf{n+m}$ indicates the constant $\mathbf{p}$ representing the

last four rules are actually *rule schemes*; e.g. $Plus\,\mathbf{n}\,\mathbf{m} \to \mathbf{n}+\mathbf{m}$ stands for all reduction rules like $Plus\,\mathbf{0}\,\mathbf{0} \to \mathbf{0}$, $Plus\,\mathbf{0}\,\mathbf{1} \to \mathbf{1}, \ldots,$ $Plus\,\mathbf{37}\,\mathbf{63} \to \mathbf{100}, \ldots$.

3.2.21. EXERCISE. Show that in the TRS SKIM a term *fac* can be defined that represents the factorial function $!: \mathbb{N} \to \mathbb{N}$, with $0! = 1$ and $(n+1)! = n! \times (n+1)$. (Hint: define some auxiliary functions, especially the predecessor function $n \mapsto n-1$.)

Another aspect of CL, contributing to its status of being a TRS that is simple to define, but infinitely rich in structure, is that it has strong undecidability properties. For example it is undecidable whether two terms are convertible, and also whether a term has a normal form. More details on this issue can be found in Section 5.4.

3.2.4. Mixed notation

The applicative notation of CL can be mixed with the usual *functional* form, as in CL with test for syntactical equality in Table 3.5. However, some care

$Sxyz$	$\to$	$xz(yz)$
Kxy	$\to$	x
Ix	$\to$	x
$D(x,x)$	$\to$	E

Table 3.5: CL with binary equality test

should be taken. Consider the TRS in Table 3.6, where D is now a *constant* (instead of a binary operator) subject to the rewrite rule, in full notation, $Ap(Ap(D,x),x) \to E$. These two TRSs have very different properties. As we shall see later, in Section 5.6, the TRS of Table 3.5 is confluent by Toyama's Theorem on the modularity of confluence (5.6.2). The TRS of Table 3.6, here denoted by CL-e, is not, as was shown in Klop [1980]. This result can also be found in Barendregt [1984].

$Sxyz$	$\to$	$xz(yz)$
Kxy	$\to$	x
Ix	$\to$	x
Dxx	$\to$	E

Table 3.6: CL with applicative equality test (CL-e)

natural number $p = n + m$. Likewise we use $\mathbf{n} \cdot \mathbf{m}$.

3.2.22. Remarks. (i) There is a connexion between the non-confluence of CL with applicative equality test (CL-e) and of the TRS that we saw earlier in Exercise 2.7.20. As a matter of fact, Klop [1980] shows that its applicative variant $\{Dxx \to E, Cx \to Dx(Cx), A \to CA\}$ is definable in CL-e. This explains (although the actual proof is more involved) why the TRS CL-e is not CR.

(ii) Although CL-e lacks the Church–Rosser property, it satifies UN, uniqueness of normal forms.

3.2.23. Exercise (Klop [1985]). We give two alternative formulations, FP and FP$'$, of the existence of fixed points in CL. We also give an alternative formulation, CC, of the property of combinatorial completeness. Note that these notions make sense not only for CL, but also for extensions of CL.

FP: For every term F there exists a term X such that $X \twoheadrightarrow FX$.

FP$'$: For every one-hole context $C[\,]$ there exists a term X such that $X \twoheadrightarrow C[X]$.

CC: If C is an arbitrary context with n holes, then there exists a term F such that

$$Fx_1 \dots x_n \twoheadrightarrow C[x_1, \dots, x_n]$$

(i) Show that for CL we have both FP and FP$'$.

(ii) Consider the rewriting system $\mathcal{D}$ of Table 3.5. Show that $\mathcal{D}$ satisfies FP, but does not satisfy FP$'$. (Hint: for a counterexample to FP$'$ consider the context $C[\,] = D(\Box, I)$ and show that in a $\mathcal{D}$-reduction there does not occur arbitrarily deep nesting of terms of the form $D(t, s)$.)

(iii) Show that CC $\Rightarrow$ (FP $\Leftrightarrow$ FP$'$).

(iv) Conclude that $\mathcal{D}$ does not have the property CC.

So, adding the so-called *discriminator* D in functional form to CL, as in Table 3.5, destroys combinatorial completeness (see Theorem 3.2.9). On the other hand, as will be seen in Section 5.6, by Toyama's Modularity Theorem confluence is preserved by this extension. Adding the discriminator D in applicative form, as in CL-e of Table 3.6, has just the opposite effect: it preserves combinatorial completeness, but destroys confluence.

3.2.5. Currying

In the practice of functional programming one often writes terms in the applicative fashion of CL, for example $A(x, S(y))$ would be rendered as $Ax(Sy)$. In this expression the binary function symbol for application Ap is hidden by the notation conventions of CL, which we will assume throughout.

We use the notion of an *applicative TRS*, by which we simply mean a TRS (Σ, R), where Σ consists of one binary function symbol Ap and for the rest only constant symbols. Typical examples are CL and the SKIM. (As a matter of fact, we could be a bit more restrictive, and anly count a TRS as applicative if it can be obtained by the process of currying, as defined below.)

We call the transformation of a first-order term rewriting system to its applicative version *currying*.[5]

3.2.24. DEFINITION. We define the *curried* version $(\Sigma, R)^{cur} = (\Sigma^{cur},\ R^{cur})$ of a TRS (Σ, R) as follows.

(i) The signature Σ^{cur} has the binary function symbol Ap for application, the constants from Σ, and for every function symbol F of positive arity a new constant c_F, also rendered as F.

(ii) The rule set R^{cur} contains all rules of the form $cur(t) \to cur(s)$ for $t \to s$ in R, where cur is inductively defined by

$$\begin{array}{lcl} cur(x) & = & x \\ cur(F(t_1, \ldots, t_n)) & = & Ft_1 \ldots t_n \end{array}$$

In this definition we assume the signatures Σ and Σ^{cur} to be disjoint. So in particular Σ should not already contain the symbol for application Ap.

3.2.25. EXAMPLE. If we start out with $R = \{A(x,0) \to x,\ A(x,S((y)) \to S(A(x,y))\}$, then this definition yields the set of curried rules $R^{cur} = \{Ax0 \to x,\ Ax(Sy) \to S(Axy)\}$.

A natural question now is how properties of (Σ, R) are preserved after currying. Regarding the most important properties, CR and SN, it turns out that both are preserved. For CR this was proved in Kahrs [1995]. Proofs of the preservation of SN under *cur* can be found in Kennaway et al. [1995] and in Middeldorp et al. [1996]. We will not cover these proofs here.

3.2.26. EXERCISE. Prove that the converse implications hold as well.

3.3. Special rewriting formats

3.3.1. Ground TRSs

A TRS of which the reduction rules consist of only closed terms will be called a *ground TRS*. Variables do not occur in the reductions in a ground TRS, so they may as well be forgotten in this case – although formally they are there, as solitary normal forms.

Also in an arbitrary TRS $\mathcal{R} = (\Sigma, R)$ we sometimes want to restrict our attention to reductions from closed terms. Since the set of closed terms $Ter_0(\Sigma)$ is closed under reduction, the system $(\mathcal{R})_0 = (Ter_0(\Sigma), \to)$ can be considered as a *sub-TRS* of $\mathcal{R}$, in the sense of the notion of sub-ARS of Definition 1.1.6. We call $(\mathcal{R})_0$ the *closed fragment* of (Σ, R). Note that we, somewhat sloppily, denoted the restriction of $\to$ to $Ter_0(\Sigma)$ also by $\to$. If the

[5]This refers to H.B. Curry, although the name of Schönfinkel could also have been attached here, see the beginning of this section.

signature Σ has no constants, then the set $Ter_0(\Sigma)$ will be empty; therefore we will not consider that case.

Obviously, a TRS is ground-CR (ground-WN, ground-SN, etc.) precisely when its closed fragment is CR (WN, SN, etc.). We have that CR implies ground-CR, and likewise for other properties such as UN, SN, WN, etc. But the converse does not hold in all cases. The following example illustrates that WN and ground-WN do not necessarily coincide.

3.3.1. EXAMPLE. Extend the TRS $\mathcal{N}^{AM} = (\Sigma, R)$ of Example 2.3.1 to (Σ, R'), where $R' = R \cup \{A(x, y) \to A(y, x)\}$, the extra rule expressing commutativity of addition. The reduction graph of the term $A(0, S(0))$ is depicted in Figure 3.2.

Now (Σ, R') is not WN: the term $A(x, y)$ has no normal form. However, $(\Sigma, R')_0$ is WN, as will be proved in Exercise 3.3.2.

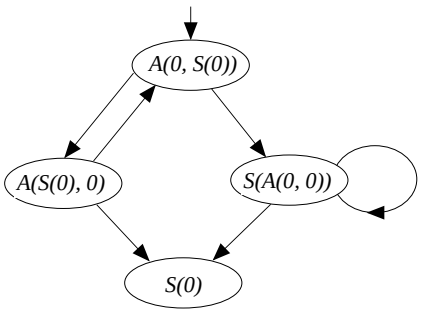


Figure 3.2: Reduction graph of $A(0, S(0))$

3.3.2. EXERCISE. Consider the TRS $\mathcal{N}^{AM}$ from Example 2.3.1. Show that the extension of $\mathcal{N}^{AM}$ with the reduction rule $A(x, y) \to A(y, x)$ from Example 3.3.1 is indeed ground-WN. (Note that the extended TRS is not ground-SN.)

3.3.3. EXERCISE. Consider the TRS $\mathcal{R}$ with unary function symbol F and constant symbol 0 and with the rewrite rules $F(x) \to x, F(x) \to 0, 0 \to 0$.

(i) Does $\mathcal{R}$ have the property UN? And UN$^{\to}$?

(ii) The same questions for $(\mathcal{R})_0$.

3.3.4. EXAMPLE (elementary school arithmetic). The (ground) TRS $\mathcal{A}$ has two binary function symbols, *Plus* and *Times*, and in addition infinitely many constant symbols: there is a constant $\mathbf{n}$ for each $n \in \mathbb{N}$. There are also infinitely many reduction rules, namely all reduction rules like $Plus(\mathbf{0}, \mathbf{0}) \to \mathbf{0}$, $Plus(\mathbf{0}, \mathbf{1}) \to \mathbf{1}, \ldots, Plus(\mathbf{37}, \mathbf{63}) \to \mathbf{100}, \ldots$. In schematic form:

$$\begin{aligned} Plus(\mathbf{n}, \mathbf{m}) &\to \mathbf{n} + \mathbf{m} \\ Times(\mathbf{n}, \mathbf{m}) &\to \mathbf{n} \cdot \mathbf{m} \end{aligned}$$

3.3.5. EXERCISE. Consider the TRS A, expressing elementary school arithmetic, of Example 3.3.4.

(i) Prove that $\mathcal{A}$ is SN.

(ii) What are the normal forms of $\mathcal{A}$?

(iii) Prove that $\mathcal{A}$ is UN$^{\rightarrow}$, i.e., each term reduces to at most one normal form.

3.3.2. Constructor term rewriting systems

A very useful subclass of TRSs is that of the *constructor TRSs*. A constructor TRS is a term rewriting system in which the set of function symbols can be partitioned into a set $\mathcal{D}$ of *defined function symbols* and a set $\mathcal{C}$ of constructor symbols or *constructors*, such that for every rewrite rule $t \to s$, the left-hand side t has the form $F(t_1, \ldots, t_n)$ with $F \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathit{Ter}(\mathcal{C}, \mathit{Var})$, the set of terms built from variables and constructors.

3.3.6. Example. (i) The TRS $\mathcal{N}^{AM}$ of Example 2.3.1 is a constructor TRS with $\mathcal{D} = \{A, M\}$ and $\mathcal{C} = \{0, S\}$.

(ii) Combinatory logic, the key example, discussed extensively in Section 3.2 (for the rules see Table 3.1) is not a constructor TRS: e.g. in the left-hand side of the second rule, $Ap(Ap(K, x), y)$, the head-symbol Ap reappears in one of the arguments.

(iii) The TRS $\mathcal{R}_M$ that will in Subsection 5.3.1 be associated with a Turing machine M is a constructor TRS with constructors $\triangleright, \square, a, b, \ldots$ and defined symbols the $q \in Q$.

(iv) Recursive program schemes (RPSs) from Definition 3.3.7 are constructor TRSs.

Recursive program schemes

3.3.7. Definition. A *recursive program scheme* (RPS) is a TRS $\mathcal{R} = (\Sigma, R)$, satisfying the following.

(i) The signature Σ can be divided into two disjoint sets of function symbols $\mathcal{F} = \{F_1, \ldots, F_n\}$ (the *unknown* functions) and $\mathcal{G} = \{G_1, \ldots, G_k\}$ (the *known* or *basic* functions).

(ii) The reduction rules of $\mathcal{R}$ have the form

$$F_i(x_1, \ldots, x_{m_i}) \to t_i$$

where all the displayed variables are pairwise different. For each F_i in $\mathcal{F}$ there is exactly one rule.

3.3.8. Example.

$$\begin{array}{lcl} F_1(x) & \to & G_1(x, F_1(x), F_2(x, x)) \\ F_2(x, y) & \to & G_2(F_2(x, x), F_1(G_3)) \end{array}$$

We will see in Chapter 4 that every RPS is orthogonal, and hence confluent. Another interesting fact about RPSs is the following.

3.3.9. THEOREM (Khasidashvili [1993b, 1993a]). *For RPSs, the properties SN and WN are decidable.*

For an extensive treatise on semantical aspects of recursive program schemes, see Courcelle [1990], where RPSs are called 'Recursive Applicative Program Schemes'.

3.3.3. Many-sorted term rewriting systems

Term rewriting systems (Σ, R) as defined here are sometimes called *one-sorted*, as they correspond to algebraic data type specifications (by replacing '$\to$' by '$=$' in R) where the signature Σ has just one sort (which therefore was not mentioned). It is straightforward to extend our previous definitions to the *many-sorted* case. The definition of term formation is as usual in many-sorted abstract data type specifications. What it amounts to roughly is that, given a set of sorts (or 'types'), in addition to specifying an arity for each function symbol in the signature, also sort information needs to be supplied. That is, if f is an n-ary function symbol, n sorts S_i have to be specified for the respective argument places, as well as an 'end' sort S. Then $f(t_1, \ldots, t_n)$ will be a well-formed term only if for each i, $1 \leq i \leq n$, t_i has sort S_i; the sort of $f(t_1, \ldots, t_n)$ is S. We will in this book stick to one-sorted TRSs, just noting that much of the theory extends easily to the many-sorted case.

For an extensive treatment of many-sorted specifications and rewriting we refer to Ehrig and Mahr [1985] and Drosten [1989].

3.3.4. String rewriting systems

String rewriting systems, also called *semi-Thue systems*, were introduced as an instrument for deciding word problems in semi-groups. They have been extensively studied from the perspective of term rewriting, e.g. in Jantzen [1988], Book [1987], Avenhaus [1995].

Consider a finite set $\Sigma = \{a_1, \ldots, a_n\}$ of symbols, called the *alphabet*. The set Σ^* of *words* or *strings* over Σ consists of the finite sequences of elements from the alphabet, including the empty sequence, denoted by Λ. The relevant binary operation on strings is concatenation. If v, w are words, their concatenation can be denoted by $v \cdot w$, or, shorter, just vw.

Now given a set E of string equations, the question is to decide for arbitrary words w_1, w_2 whether they can be proved equal. (This is the *word problem* in its original form.) Here term rewriting techniques can be used: transform the equations into rewrite rules, preferably by just giving them an appropriate orientation. The resulting system is what is called a *string rewriting system*. If it is complete, then, as we have seen in Section 2.6, we have a decision procedure for the word problem: compute and compare the respective normal forms of the strings w_1, w_2. In general a string rewriting system will be given by the ordered pairs of words that constitute its rewrite rules.

Note that, even though it is close to rewriting as we defined it before, a string rewriting system does not have the signature of a TRS right away. Concatenation is associative, and there is no syntactic operation on first-order terms that shares this behaviour. However, by means of a proper translation a string rewriting system can be viewed as a TRS. There are several ways to do this. We demonstrate one very natural way in the following example.

3.3.10. EXAMPLE. Let $\mathcal{T} = \{(aba, bab)\}$ be a one-rule string rewriting system. Then $\mathcal{T}$ corresponds to the TRS $\mathcal{R}$ with unary function symbols a, b and a constant o, and the reduction rule $a(b(a(x))) \to b(a(b(x)))$. Now a reduction step in $\mathcal{T}$, for example $bbabaaa \to bbbabaa$, translates in $\mathcal{R}$ to the reduction step $b(b(a(b(a(a(a(o))))))) \to b(b(b(a(b(a(a(o))))))$. It is easy to see that this translation gives an 'isomorphism' between $\mathcal{T}$ and $(\mathcal{R})_0$, the restriction of $\mathcal{R}$ to ground terms.

A second way to let a string rewriting system correspond to a TRS is by introducing an associative concatenation operator, and letting the symbols of the string rewriting system correspond to constant symbols in the TRS. In fact, a natural correspondence in this way requires that one works with term rewriting modulo an equivalence. (See e.g. Bachmair and Plaisted [1985] or Plaisted [1985].)

Historically related to semi-Thue systems is the notion of a *Post canonical system*, named after the logician Emil Post. As a matter of fact, Post canonical systems were one specimen in a larger class of such systems that were studied as a general framework for deductive systems of logic, in the 1920s and 1930s, in the style of what are now called Hilbert (axiomatic) deduction systems. A Post canonical system also rewrites strings, but now these are allowed to contain variables. There are a set of axioms and a set of so-called *production rules* of the form $\alpha_1, \ldots, \alpha_n \to \alpha$; here $\alpha_1, \ldots, \alpha_n$ are called the *premises* and α the *conclusion* of a rule. Starting with the axioms, the production rules can be used to produce new strings: if the premises $\alpha_1, \ldots, \alpha_n$ have been produced already, then the conclusion α may be added.

Notable here is the quite different focus that goes with this use of the concept of term rewriting: the interest is not in termination, confluence or uniqueness of normal forms. The typical question is now whether a certain string can be produced from the axioms, and in general, to characterize the set of strings that can be so produced. Translated into the language of TRSs: which terms are in the reduction graph of a given one? Anyway, TRS techniques may in principle be used. And moreover, translations like the ones given in this section sometimes make it possible to transfer e.g. (un)decidability results from one type of system to another.

3.3.11. EXERCISE (Hofstadter's MU-puzzle). Hofstadter [1979] presented the following nice puzzle, in order to clarify the notion of a formal system. Consider the deductive system, in the form of a Post canonical system, of Table 3.7.

Axiom:

$$MI$$

Production rules:

$$\begin{array}{rcl} xI & \to & xIU \\ Mx & \to & Mxx \\ xIIIy & \to & xUy \\ xUUy & \to & xy \end{array}$$

Table 3.7: The production rules of Hofstadter's MU-puzzle

(i) Is the word MU derivable in this system?

(ii) Translate the system into a TRS. More precisely, give a TRS $\mathcal{R}$ and an injective function ϕ from the words over the alphabet $\{M, I, U\}$ to $Ter(\mathcal{R})$ in such a way that we get $MI \vdash A \Leftrightarrow \phi(MI) \twoheadrightarrow \phi(A)$.

3.4. Conditional term rewriting systems

In this section we discuss another extension of the term rewriting format, namely with conditions. Conditional rewriting has its roots in universal algebra (Meinke and Tucker [1991]) and in the field of algebraic specifications (Ehrig and Mahr [1985]). Moreover, conditional term rewriting systems play a role in the integration of the disciplines of functional programming and logic programming; see e.g. Dershowitz and Okada [1990] and Hanus [1997, 2000][6].

Before we turn to conditional rewriting proper, we briefly discuss equational systems with conditions.

3.4.1. Conditional specifications

From an algebraic point of view it is quite natural to consider conditional equations. This is nicely illustrated in Table 3.8 by the specification $\mathcal{I}$ of the integers with a unary function symbol *pos*, with $pos(t)$ evaluating to *true* if and only if t represents a positive integer. Baader and Nipkow [1998] point out that without extending $\mathcal{I}$'s signature, $0, P, S, true, false, pos$, it is not possible to give an equivalent specification using only finitely many *unconditional* equations.

In rendering conditional equations we conform to the notation which is normally used in equational logic programming, writing the conditions at the right-hand side of an inverse implication. (See e.g. Apt [1990].) The general

[6]Conditions figure as *constraints* in the integrated 'functional logic programming language' Curry.

$P(S(x))$	$=$	x				
$S(P(x))$	$=$	x				
$pos(S(0))$	$=$	$true$				
$pos(0)$	$=$	$false$				
$pos(S(x))$	$=$	$true$	$\Leftarrow$	$pos(x)$	$=$	$true$
$pos(P(x))$	$=$	$false$	$\Leftarrow$	$pos(x)$	$=$	$false$

Table 3.8: Conditional specification of the integers

form of a conditional equation is then

$$t = s \quad \Leftarrow \quad t_1 = s_1, \ldots, t_n = s_n$$

Here the equation on the left, $t = s$, might be called the *conclusion* of the conditional equation; the $t_i = s_i$ are the *conditions*. Also reasoning with such conditional equations is quite natural. The conclusion of a conditional equation with substitution σ can be derived when the instances of its conditions that result from σ have been derived earlier.

We give some more examples of conditional specifications.

3.4.1. Examples. (i) A specification using conditional equations of *gcd*, computing the greatest common divisor on natural numbers with 0 and successor S, is given in Table 3.9. To keep the specification one-sorted, 0 and $S(0)$ could be used as booleans *false* and *true* respectively. Furthermore, '$-$' is cut-off subtraction.

$0 < 0$	$=$	$false$		$S(x) - S(y)$	$=$	$x - y$
$0 < S(x)$	$=$	$true$		$0 - x$	$=$	0
$S(x) < 0$	$=$	$false$		$x - 0$	$=$	x
$S(x) < S(y)$	$=$	$x < y$				
$gcd(x, y)$	$=$	$gcd(x - y, y)$	$\Leftarrow$	$y < x = true$		
$gcd(x, y)$	$=$	$gcd(x, y - x)$	$\Leftarrow$	$x < y = true$		
$gcd(x, x)$	$=$	x				

Table 3.9: Specification of *gcd* with conditional equations

(ii) A natural rule for the transitivity of a less than relation, here represented by a binary function symbol L, could be the following.

$$L(x, z) = true \quad \Leftarrow \quad L(x, y) = true \wedge L(y, z) = true$$

Note that here the variable y, occurring in the conditions, but not in the consequent, gets existential meaning: $L(x, z) = true$ will hold if *some* y can be given such that $L(x, y) = true$ and $L(y, z) = true$.

3.4.2. Conditional rewriting

A *conditional term rewriting system* (CTRS) $\mathcal{C}$ will consist of a first-order signature Σ with a set of conditional rewrite rules over Σ. The terms of $\mathcal{C}$ are just the usual first-order terms over Σ. We first specify the general format of a *conditional rewrite rule*, leaving the form of the conditions yet unspecified.

3.4.2. Definition. (i) A *conditional rewrite rule* has the form

$$\rho: \qquad t \to s \;\Leftarrow\; C_1, \ldots, C_n$$

Here $C_1, \ldots, C_n$ are the conditions, $n \geq 0$. We require that the rule that remains when ρ is stripped of its conditions is a rewrite rule in the usual, unconditional, sense. We will call this underlying unconditional rule ρ_u.

$$\rho_u: \qquad t \to s$$

(ii) The TRS that results from a CTRS $\mathcal{C}$ by stripping all its reduction rules of their conditions will be denoted by $\mathcal{C}_u$.

Note that we may have $n = 0$ in (i). Then ρ has no conditions (and hence $\rho = \rho_u$). This makes the unconditional rewrite rules a borderline case of the conditional ones.

Now we turn to the conditions. Let us take the algebraic examples above as a starting point. If we just leave the conditions in the form of equations, the result is a so-called *semi-equational* CTRS. There are other possibilities for casting the conditions, however, giving rise to different versions of the notion of CTRS. Three natural ones are given in Definition 3.4.3. They were proposed by Dershowitz, Okada and Sivakumar [1988], thereby extending a classification introduced in Bergstra and Klop [1986].

We employ the following notation. Given a rewrite relation $\to$, two terms t, s are called *joinable*, notation $t \downarrow s$, if $t \twoheadrightarrow u$ and $s \twoheadrightarrow u$ for some term u.

3.4.3. Definition. We distinguish three special types of CTRS, with the format of the rewrite rules as displayed:

(i) *semi-equational* systems

$$t_0 \to s_0 \;\Leftarrow\; t_1 = s_1, \ldots, t_n = s_n$$

(ii) *join* systems

$$t_0 \to s_0 \;\Leftarrow\; t_1 \downarrow s_1, \ldots, t_n \downarrow s_n$$

(iii) *normal* systems

$$t_0 \to s_0 \;\Leftarrow\; t_1 \twoheadrightarrow n_1, \ldots, t_k \twoheadrightarrow n_k$$

(with $n_1, \ldots, n_k$ ground normal forms with respect to the unconditional system $\mathcal{C}_u$).

Note that the normal systems are a special case of the join systems, since, when n is a ground normal form, the conditions $t \twoheadrightarrow n$ and $t \downarrow n$ are equivalent.

In each of these three cases the definition of $\rightarrow$ depends on conditions involving a reference to $\rightarrow$ itself (via $=$, $\downarrow$ or $\twoheadrightarrow$). The conditions can then not be evaluated independently of the rewrite relation that is being defined. The rewrite rules should therefore be taken as constituting an inductive definition of $\rightarrow$, which is possible since the conditions make only positive reference to $\rightarrow$. This is made precise in Definition 3.4.5.

But first we will handle the simpler case where the conditions have a fixed meaning, that is independent of the reduction relation being defined. We will speak in this case of a *generalized* CTRS.

3.4.4. Definition. (i) In a *generalized* CTRS the reduction rules are of the form

$$\rho: \qquad t \rightarrow s \;\Leftarrow\; P_1(x_1, \ldots, x_m), \ldots, P_n(x_1, \ldots, x_m)$$

where the P_i are any fixed predicates on terms.

(ii) The *instances* of the conditional rule ρ are exactly those instances $t^\sigma \rightarrow s^\sigma$ of ρ_u that result from a substitution σ such that the conditions $P_1(\sigma(x_1), \ldots, \sigma(x_m)), \ldots, P_n(\sigma(x_1), \ldots, (\sigma(x_m))$ hold.

(iii) A rewrite step of $\mathcal{C}$ consists of contracting a redex, i.e. an instance of a conditional rewriting rule, in an arbitrary context:

$$C[t^\sigma] \rightarrow C[s^\sigma]$$

Note that the rewriting steps of a generalized CTRS are defined just as in a TRS, with the restriction that the conditions of the corresponding conditional rule have to be satisfied. Now all common TRS notions and notations can be used for CTRSs. In particular we have clear notions of reduction ($\twoheadrightarrow$) and conversion ($=$). We already observed that unconditional reduction rules can be considered as a special case of conditional ones, so the usual notion of TRS can be considered a special case of a CTRS.

Now returning to the inductive notions of CTRS of Definition 3.4.3, we proceed by defining $\rightarrow$ as the union of an increasing sequence of intermediate rewrite relations $\rightarrow_n$. Here $\rightarrow_{n+1}$ is the reduction relation of a generalized CTRS $\mathcal{C}_{n+1}$, with conditions referring to $\rightarrow_n$ of $\mathcal{C}_n$, that was defined at an earlier stage.

3.4.5. Definition. (i) Let $\mathcal{C}$ be a normal CTRS with rewrite rules

$$\rho_j : t_j \rightarrow s_j \Leftarrow t_{j,1} \twoheadrightarrow n_{j,1}, \ldots, t_{j,k_j} \twoheadrightarrow n_{j,k_j}$$

($j = 1, \ldots, m$). The $n_{j,1}, \ldots, n_{j,k_j}$ are normal forms with respect to the unconditional part $\mathcal{C}_u$.

We define, by induction on n, a sequence $\mathcal{C}_n$ ($n \geq 0$) of generalized CTRSs. The reduction relation of $\mathcal{C}_n$ is denoted by $\to_n$.

- *Basis.* The CTRS $\mathcal{C}_0$ has no reduction rules. In other words, we start with the empty reduction relation: $\to_0 = \varnothing$. Note that as a result we have $\twoheadrightarrow_0 = \equiv$.
- *Induction step.* Suppose that the CTRS $\mathcal{C}_p$ with reduction relation $\to_p$ (and thereby also $\twoheadrightarrow_p$) is defined. Then $\mathcal{C}_{p+1}$ is the generalized CTRS with the reduction rules

$$\rho_{p+1,j}: \quad t_j \to s_j \;\Leftarrow\; t_{j,1} \twoheadrightarrow_p n_{j,1},\ \ldots,\ t_{j,k_j} \twoheadrightarrow_p n_{j,k_j}$$

  $(j = 1, \ldots, m)$.

Then, every $\mathcal{C}_n$ with $n \geq 0$ being defined, we take the reduction relation $\to$ of $\mathcal{C}$ to be the union: $\to = \bigcup_{n\geq 0} \to_n$.

(ii) The rewrite relations of semi-equational and join CTRSs are defined analogously.

In the following exercise it is shown that the relations $\to_n$ form an increasing sequence and, moreover, that the final relation $\to$ is a fixed point of the construction. The upshot is that, after all, the conditions in the rules defining $\to$ *can* be taken to refer to $\to$ itself. Now we make the case of a semi-equational CTRS explicit.

3.4.6. Exercise. (i) Show that for each n we have $\to_n \subseteq \to_{n+1}$.

(ii) Consider a semi-equational CTRS $\mathcal{C}$ with reduction relation $\to$ (and conversion $=$). Define $\mathcal{C}'$ with $\to'$ as the generalized CTRS with for each reduction rule

$$\rho: \quad t \to s \;\Leftarrow\; t_1 = s_1, \ldots, t_k = s_k$$

of $\mathcal{C}$ the reduction rule

$$\rho': \quad t \to' s \;\Leftarrow\; t_1 = s_1, \ldots, t_k = s_k$$

Show that the relations $\to$ and $\to'$ coincide. Conclude that also conversion ($=$) in $\mathcal{C}$ and $\mathcal{C}'$ are the same.

(iii) Prove the analogous results for normal and join CTRSs.

One must remain careful on how the conditions are interpreted.

3.4.7. Remark. Incorporating negative conditions containing $\to$ in a CTRS would disturb the inductive definition. A simple example already illustrates the point. Consider the CTRS consisting of the single conditional rewrite rule

$$a \to b \;\Leftarrow\; a \neq b$$

Does $a \to b$ hold? If not, then yes by the conditional rule. If yes, then by which reduction rule?

For this reason the conditions of normal systems cannot be put in the form $t \twoheadrightarrow_! n$: t reduces to n and n is a normal form *with respect to the relation* $\rightarrow$ *being defined.* Indeed, this type of condition would have a hidden negative part: n does not reduce. Consider for example the problematic single-rule CTRS

$$a \rightarrow a \quad \Leftarrow \quad a \twoheadrightarrow_! a$$

Does $a \rightarrow a$ hold?

Allowing conditions of the form $t \twoheadrightarrow s$ without requiring s to be a normal form at all is also not very attractive. The conditions would in general be unstable under reduction, even if the reduction relation corresponding to the CTRS turned out to be confluent.

3.4.8. Remark. In a rewrite rule $t \rightarrow s$ one requires that in s no new variables appear with respect to t. The same requirement is made for conditional rewrite rules $t \rightarrow s \Leftarrow C$. But, as observed in Dershowitz, Okada and Sivakumar [1988], for CTRSs it would make good sense to lift this requirement, as e.g. in the following perfectly natural conditional rewrite specification of the Fibonacci numbers:

$$\begin{array}{lcll} Fib(0) & \rightarrow & \langle 0,1\rangle & \\ Fib(x+1) & \rightarrow & \langle z, y+z\rangle & \Leftarrow \quad Fib(x) \downarrow \langle y,z\rangle \end{array}$$

We will not study this more liberal format here, although it is quite natural. See e.g. Suzuki et al. [1995] and Yamada et al. [2000].

Whereas in the unconditional case it is easy to decide whether a term is in normal form, this may change in the case of CTRSs. Determining a redex is no longer merely a matter of matching against the left-hand sides of reduction rules; conditions may also have to be checked. As a matter of fact, it turns out that there are semi-equational, normal and join CTRSs for which the set of normal forms is undecidable.

4

Orthogonality

Jan Willem Klop
Vincent van Oostrom
Roel de Vrijer

This chapter treats the basic theory of orthogonal (first-order) term rewriting systems. After some examples of orthogonal and weakly orthogonal TRSs, we prove confluence for orthogonal TRSs in several ways. Key concepts in the confluence proofs are: descendant and residual (Section 4.2), parallel move (Section 4.3), labelling (Section 4.2), underlining (Section 4.4), complete development and finiteness of developments (Section 4.5), tiling with elementary diagrams (Section 4.6), multi-step (Section 4.5). In the section about non-erasing reductions the theorems of Church and O'Donnell are proved. Next, a brief introduction to reduction strategies is given, with definitions of some important strategies and several notions of fairness; an extensive treatment can be found in Chapter 9 of Terese [2003].

4.1. Introduction

In Section 2.7 two potential sources of non-confluence have been identified. The first one is the phenomenon of overlapping reduction rules, leading to critical pairs. The second source is non-left-linearity (as illustrated by Exercise 2.7.20). In this chapter we will restrict our attention to TRSs where these two sources have been eliminated. The effects of the reduction rules then appear to be *orthogonal* to each other, in the sense that they can no longer 'harmfully' interfere, i.e. in a way which might lead to non-confluence.

Below we will get back to the subjects of overlap and non-left-linearity. But first we recapture the notion of an *orthogonal* TRS in the following definition.

4.1.1. Definition. Consider a TRS $\mathcal{R}$.

(i) $\mathcal{R}$ is *non-ambiguous* (also *non-overlapping*) if there is no overlap between any of its reduction rules. Otherwise $\mathcal{R}$ is called *ambiguous*.

(ii) $\mathcal{R}$ is *left-linear* if all its reduction rules are left-linear.

(iii) $\mathcal{R}$ is *orthogonal* if it is non-ambiguous and left-linear.

Since a left-linear TRS has critical pairs precisely in those cases where some of its rules overlap, an alternative phrasing of orthogonality is *left-linear and without critical pairs*.

The requirement of having no critical pairs can be slightly weakened as follows: critical pairs are allowed, but only *trivial* ones, in the sense of consisting of identical terms. With this weakened notion of orthogonality many important results still hold, in particular confluence.

4.1.2. DEFINITION. (i) A critical pair $\langle t, s\rangle$ is *trivial* if $t \equiv s$.

(ii) A TRS $\mathcal{R}$ is *weakly orthogonal* if it is left-linear and has only trivial critical pairs.

Note that in order to verify whether a TRS is orthogonal, we have only to look at the left-hand sides of the reduction rules. For weak orthogonality we may have to check critical pairs, and then the right-hand sides also count.

Now, to get a better grasp, we dwell a little longer on the key concepts figuring in the orthogonality definitions.

Recall that *left-linearity* of a reduction rule means that it does not possess repeated variables in its left-hand side. Left-linear reduction rules were in Chapter 2 defined in Definition 2.3.3.

4.1.3. REMARK. For a left-linear rule it is only its pattern that counts in determining a redex. One problem with non-left-linear rules is that their application requires a test for syntactic equality of the arguments substituted for the variables occurring more than once. As terms may be very large, this may be laborious.

However, the main issue here is of course that the presence of non-left-linear rules may destroy the CR property. We have seen this in Exercise 2.7.20, where non-confluence was caused by the non-left-linear rule $D(x, x) \to E$. We give yet another example, making the same point.

4.1.4. EXAMPLE. (i) This example stems from Huet [1980]. Consider a TRS with the following rewrite rules.

$$\begin{array}{lcl} \infty & \to & S(\infty) \\ E(x,x) & \to & \mathit{true} \\ E(x,S(x)) & \to & \mathit{false} \end{array}$$

Note that these rules are non-overlapping. Yet confluence fails, since we have $E(\infty,\infty) \to \mathit{true}$ and also $E(\infty,\infty) \to E(\infty,S(\infty)) \to \mathit{false}$.

(ii) Let us add some interesting observations on the TRS in (i). If we add the *applicative* versions of the two rules for E to CL we obtain

$$\mathrm{CL} \cup \{Ex\,x \to \mathit{true},\ Ex(Sx) \to \mathit{false}\}$$

This TRS is also not CR, because we can define ∞ in CL using the fixed-point combinator as explained in Section 3.2.

By contrast, if we extend CL by the *functional* versions of these rules, obtaining

$$\mathrm{CL} \cup \{E(x,x) \to \mathit{true},\ E(x,S(x)) \to \mathit{false}\}$$

we *do* have CR. This can be shown using Toyama's Theorem (5.6.2); see Exercise 5.6.6. (This mix of applicative and functional notation was also discussed in Subsection 3.2.4.)

Overlap between reduction rules was explained in Definition 2.7.3 and Lemma 2.7.7 in terms of their *patterns*. Several examples of overlap were already given in Examples 2.7.5. Here is an example of a non-ambiguous TRS, so without overlap.

4.1.5. Example. Let $\mathcal{R}$ be the following TRS:

$$\begin{array}{lll} r_1: & F(G(x,S(0)),y,H(z)) & \to \quad x \\ r_2: & G(x,S(S(0))) & \to \quad 0 \\ r_3: & P(G(x,S(0))) & \to \quad S(0) \end{array}$$

E.g. the context $F(G(\Box,S(0)),\Box,H(\Box))$ is the pattern of rule r_1. We already depicted this pattern as the shaded area in Figure 2.3 in Section 2.7.

The TRS $\mathcal{R}$ has the property that *no term patterns can overlap*, i.e. $\mathcal{R}$ has the non-overlapping or non-ambiguity property. Figure 4.1 shows a term in $\mathcal{R}$ with all patterns indicated. Note how they are laid nicely apart.

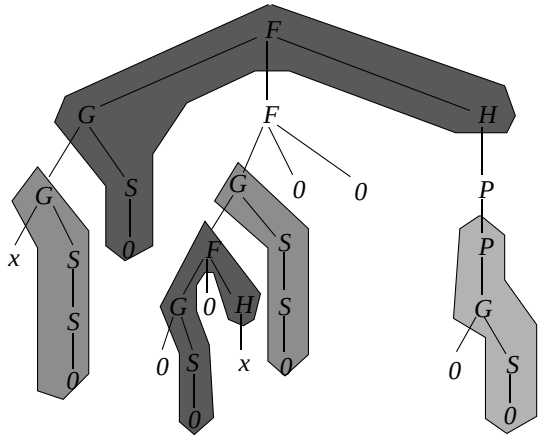


Figure 4.1: Configuration of non-overlapping rule patterns

We will now give a number of examples of orthogonal and weakly orthogonal TRSs. Later on we will have the general theorem that all such TRSs are confluent.

4.1.6. Examples. (i) *Combinatory logic* (Table 3.3) has rule patterns as in Figure 4.2; they cannot overlap. As CL is left-linear, it is therefore orthogonal.

(ii) The TRS SKIM, in Table 3.4, is orthogonal.

(iii) *Recursive program schemes*, as defined in Definition 3.3.7, are orthogonal.

(iv) Another example of an orthogonal TRS, computing the list of prime numbers, is given by the following rules, implementing the *sieve of Eratosthenes*.

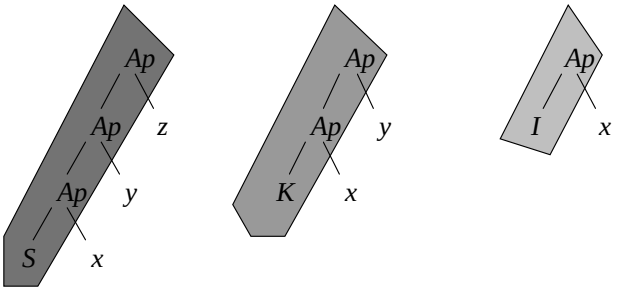


Figure 4.2: Patterns of CL-rules

$$
\begin{array}{rcl}
\mathit{eratosthenes} & \to & S(0) : \mathit{sieve}(\mathit{natsfrom}(S(S(0)))) \\
\mathit{sieve}(x : l) & \to & x : \mathit{sieve}(\mathit{filt}(x, l)) \\
\mathit{natsfrom}(x) & \to & x : \mathit{natsfrom}(S(x)) \\
\mathit{filt}(x, y : l) & \to & \mathit{test}(x, y, l, \mathit{divides}(y, x, x)) \\
\mathit{test}(x, y, l, \mathit{true}) & \to & \mathit{filt}(x, l) \\
\mathit{test}(x, y, l, \mathit{false}) & \to & y : \mathit{filt}(x, l) \\
\mathit{divides}(S(x), S(y), z) & \to & \mathit{divides}(x, y, z) \\
\mathit{divides}(0, S(y), S(z)) & \to & \mathit{false} \\
\mathit{divides}(S(x), 0, z) & \to & \mathit{divides}(S(x), z, z) \\
\mathit{divides}(0, 0, z) & \to & \mathit{true}
\end{array}
$$

(v) *Parallel-or* is a weakly orthogonal TRS.

$$
\begin{array}{lcl}
\mathit{por}(\mathit{true}, x) & \to & \mathit{true} \\
\mathit{por}(x, \mathit{true}) & \to & \mathit{true} \\
\mathit{por}(\mathit{false}, \mathit{false}) & \to & \mathit{false}
\end{array}
$$

(vi) The following two rules for *predecessor* P *and successor* S constitute a weakly orthogonal TRS.

$$
\begin{array}{lcl}
P(S(x)) & \to & x \\
S(P(x)) & \to & x
\end{array}
$$

There are two trivial critical pairs, one obtained from $P(S(P(x)))$, namely $\langle P(x), P(x)\rangle$, the other from $S(P(S(x)))$, namely $\langle S(x), S(x)\rangle$.

4.1.7. Exercise. For a deterministic Turing machine $\mathcal{M}$, the TRS $\mathcal{R}_\mathcal{M}$ as defined in Tables 5.2 and 5.3 is orthogonal. Moreover, it is a constructor TRS.

4.1.8. Exercise. In the TRS CL + δ, combinatory logic with Church's δ-rules, the signature of CL is extended by a new constant δ. By a *(*CL + δ*)-normal form* we mean a term containing no S-, K-, or I-redexes, and no closed subterms of the form δMN.

The reduction rules of CL + δ are those of CL, and in addition for each pair of closed terms M, N in (CL + δ)-normal form one of the following two reduction rules.

(1) In case $M \equiv N$ we have the rule $\delta MN \to \mathit{true}$.

(2) In case $M \not\equiv N$ we have the rule $\delta MN \to \mathit{false}$.

For *true* and *false* one may take either new constants, or the standard encoding of booleans in combinatory logic: $\mathit{true} \equiv K$, $\mathit{false} \equiv KI$.

Prove that CL + δ is an orthogonal TRS, hence confluent.

4.1.9. Exercise. Several of the following reduction rules overlap. Determine the maximal subsets of these rules that constitute a weakly orthogonal TRS: $A \to A$, $FAA \to FAA$, $FxA \to FAA$, $FAx \to FAA$, $FAx \to FxA$, $Fxy \to Fyx$.

The sections that now follow contain several proofs of confluence for orthogonal TRSs. Along with these proofs, different perspectives on orthogonality are developed, and different concepts and methods of proof, most of which will play a role in one or more of the later chapters.

The earliest confluence proof in the general setting of orthogonal first-order term rewriting is probably that of Rosen [1973]. But, as a matter of fact, earlier proofs of the confluence of CL work just as well for orthogonal TRSs in general. In the tradition of λ-calculus and combinatory logic confluence was known as the 'Church–Rosser Property'. Almost all notions in this chapter find their origin in the decade in which λ-calculus and combinatory logic were developed, starting around 1930.

4.2. Descendants and residuals

In this section we introduce the key concepts for the analysis of orthogonal rewriting: the *descendant* relation and the ensuing notion of *residual* of a redex. These notions are fundamental for all of the five or six confluence proofs in this chapter, although some use the notion of a descendant only in an implicit way. In particular, this is the case for the proof via underlinable reductions in Section 4.4. This proof could in principle be read independently of this section.

The idea of the descendant relation is simple: we want to be able to trace an element of a term t, be it a symbol, a subterm or a redex occurrence, through a reduction from t.

4.2.1. Example. Consider the TRS with rewrite rules

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

Consider the term $f(a)$, with the diverging rewrite steps $f(a) \to g(a,a)$ and $f(a) \to f(b)$. How to find a common reduct of the results $g(a,a)$ and $f(b)$? There is a simple process, which we describe now, in two steps.

(1) Determine the head symbol of the redex contracted in the first step, $f(a) \to g(a,a)$. This is the symbol f. Tracing this symbol along the second step, $f(a) \to f(b)$, we find the symbol f back in the result term $f(b)$; it is called a *descendant* of the original f. This descendant f is again the head symbol of a redex, $f(b)$. Let us contract this redex. The result is the term $g(b,b)$.

(2) Now determine the head symbol of the redex contracted in the second step, $f(a) \to f(b)$. This is the symbol a. Tracing it along the first step, $f(a) \to g(a,a)$ this time yields two copies of the symbol a in the term $g(a,a)$. Both these occurrences of a, called descendants again, are (the head symbol of) the redex a. Let us also contract these redexes. Again the result is the term $g(b,b)$.

What have we done? We found a common reduct, $g(b,b)$, by contracting the 'same' redexes, but in a different order. Not exactly the same: in $g(a,a)$ we had to contract two copies of the original a, and, moreover, the redex $f(b)$ is only a variant of the original $f(a)$, namely another instance of the same left-hand side. We call $f(b)$ and the copies of a also *descendants* (or *residuals*) of the redexes in the original term $f(a)$.

The notion of *descendant* of a symbol occurrence can be made more precise as follows. Let t be a term in an orthogonal TRS $\mathcal{R}$, and let F be a function symbol occurring in t. We will give F a marking, say by colouring it, or giving it a label, to be able to trace it during a sequence of rewrite steps. Let us label F with a superscript b, so that we get F^b. One might prefer to think of F^b as a blue symbol F. Consider the rewrite step $t \to_s t'$ that is obtained by contraction of redex s in t. Now all occurrences of F in t' which are labelled with b (or the blue ones) are called the *descendants* of the original F in t after the reduction step $t \to_s t'$. Descendants of redexes, or, for that matter, arbitrary subterms, can be found by tracing their head symbols. This works because of the one-to-one correspondence between a subterm occurrence and the occurrence of its head symbol.

We will now make this informal account precise, beginning with a definition of the *labelled* versions of a TRS.

4.2.2. Definition. Let $\mathcal{R} = (\Sigma, R)$ be a TRS and let L be a set of symbols, to be used as labels. We define the labelled TRS $\mathcal{R}^L = (\Sigma^L, R^L)$.

(i) The labelled signature Σ^L is defined as a union:

$$\Sigma^L = \Sigma \cup \{f^l \mid f \in \Sigma \text{ and } l \in L\}$$

The arity of a function symbol f^l is the same as that of f. The terms over Σ^L are the *labelled* terms.

(ii) The labelled terms can be projected back to the original unlabelled ones by removing the labels. That is, we have the projection function $|-|$ from $Ter(\Sigma^L)$ to $Ter(\Sigma)$ inductively defined by $|x| \equiv x$, $|c^l| \equiv |c| \equiv c$, $|f^l(t_1,\ldots,t_n)| \equiv |f(t_1,\ldots,t_n)| \equiv f(|t_1|,\ldots,|t_n|)$.

(iii) The set R^L of labelled rules is defined as

$$R^L = \{l \to r \mid |l| \to r \in R\}$$

We say the reduction rules have been *lifted*: in the left-hand sides arbitrary labellings of the function symbols are allowed. Notice, however, that no labels occur in the right-hand sides of the rules of R^L.

4.2.3. PROPOSITION. *Consider a left-linear TRS $\mathcal{R} = (\Sigma, R)$ and set of labels L. Let $t \in Ter(\mathcal{R})$ and let t' be a labelled term such that $|t'| \equiv t$. Then each reduction step $t \to s$ from t can in a unique way be* lifted *to a reduction step $t' \to s'$ in $\mathcal{R}^L$, with $|s'| \equiv s$. Likewise for a sequence of reduction steps in a reduction $t \twoheadrightarrow s$.*

PROOF. Obvious. For any labelling of the redex pattern of a rule in R there is a matching rule in R^L.

There is a catch though. We deliberately oversimplified a little on the aspect of uniqueness. Explicit information on the reduction rule used or the position of the contracted redex may be needed here, in case of ambiguous reduction rules or syntactic accidents. See Remark 4.2.4. □

The proposition can be strengthened to arbitrary TRSs, but then one should impose a restriction on the labelling, namely that the identical arguments of a non-left-linear redex are labelled in the same way. So, for example, a redex $f(a, a)$ with respect to the non-left-linear reduction rule $f(x, x) \to x$ should not be labelled as $f(a, a^r)$. However, for example the labelling $f(a^r, a^r)$ would be fine.

4.2.4. REMARK. Syntactic accidents were indicated in Example 2.2.9. We gave the example of a reduction rule $I(x) \to x$. Consider the step $I(I(x)) \to I(x)$. Suppose we want to lift it to a step from the labelled term $I(I^\ell(x))$. How this should be done depends on which redex was contracted. If the redex was $I(I(x))$, then we have the step $I(I(x)) \to_{I(I(x))} I(x)$, which lifts to $I(I^\ell(x)) \to I^\ell(x)$. In contrast, the step $I(I(x)) \to_{I(x)} I(x)$ lifts to $I(I^\ell(x)) \to I(x)$. Similar examples can be given for cases of overlap.

To guarantee uniqueness under all circumstances the original step should be fully rendered as $t \to_{\rho,r} s$ where ρ is the reduction rule used, and r the contracted redex occurrence.

For expository reasons we will in this chapter not always be fully accurate in similar matters.

4.2.5. EXAMPLE. Consider the rewrite rule $f(a, x) \to f(f(x, x), a)$ and the reduction step $f(f(a, a), a) \to f(f(f(a, a), a), a)$. We want to trace the three as in the original term. We can do this by giving them each a different colour, say blue, red and green, here represented by the labels b, r, g, and lifting the reduction step: $f(f(a^b, a^r), a^g) \to f(f(f(a^r, a^r), a), a^g)$. We see that the blue a, belonging to the pattern of the contracted redex, has no descendant. The green a, which occurs

disjoint with the redex, is just copied once. The red a, which happens to be the redex argument, gets multiplied. It is also interesting to note that one a in the reduct has no label: it does not descend from one of the original as.

We now give a formal definition of the descendant relation. We will use only one label, to trace one symbol at a time. But, equivalently, we could have labelled more symbols at once, as in the example. We only have to make sure then that we start with an *initial labelling*, i.e. one satisfying the requirement that different symbol occurrences get different labels. Otherwise we cannot distinguish between descendants of symbols carrying the same label.

4.2.6. Definition. (i) Let t be a term in a TRS $\mathcal{R}$, and let s be a redex and F a symbol occurrence in t. Let t_F be the term that results from t by labelling (only) the symbol occurrence F, say with label ℓ. Then the reduction step $t \to_s t'$ can according to Proposition 4.2.3 be *lifted* to a reduction step $t_F \to t''$ in the TRS $\mathcal{R}^{\{\ell\}}$. The occurrences of F in t' that have label ℓ in t'' are the *descendants* of the original symbol occurrence F in t. Conversely, the original F is called the *ancestor* of its descendants.

(ii) The descendant/ancestor relation is extended to subterm occurrences via their head symbols. So to find the descendants of subterm u of t just label its head symbol and lift. Then the subterms of t' with their head symbol labelled in t'' are the descendants of u.

(iii) Since a redex is a special case of a subterm, we have hereby also defined what are the descendants of a redex.

(iv) The descendant/ancestor relation is extended over a sequence of rewrite steps

$$t \to_{s'} t' \to_{s''} t'' \to \cdots \to t^{(n-1)} \to_{s^{(n)}} t^{(n)}$$

simply by transitivity. Put concisely: descendants of descendants are descendants.

Unlike descendants, ancestors are always unique, as one easily verifies.

Note that the above definition makes sense for all TRSs, not just for the orthogonal ones. What distinguishes orthogonal rewriting is the fact that descendants of redexes will always be redexes again. Moreover, contraction of the descendants of one step after the other or vice versa results in the same end term. As a matter of fact, this observation is also not completely restricted to orthogonal TRSs; it holds for any pair of *orthogonal reduction steps*.

4.2.7. Definition. Consider in a term t two non-overlapping redexes s_0, s_1, according to left-linear reduction rules. Then the rewrite steps $t \to_{s_0} t_0$ and $t \to_{s_1} t_1$ that result from contracting, respectively, s_0 and s_1 are called *orthogonal steps*.

The proof of the following important proposition, which might be considered as the key property of orthogonal rewriting, makes use of the proof of the Critical Pair Lemma (2.7.15). It is not difficult, but it may need some concentration. One might prefer to first have a look at Remark 4.2.11 and Example 4.2.12.

4.2.8. PROPOSITION. *Consider two orthogonal steps $t \to_{s_0} t_0$ and $t \to_{s_1} t_1$. Then the descendants of s_1 in t_0 after the reduction step $t \to_{s_0} t_0$ are again redexes. Moreover, a common reduct of t_0 and t_1 can be reached from t_0 by contracting the descendants of s_1 and from t_1 by contracting the descendants of s_0.*[1]

PROOF. Since there is no overlap, the pattern of s_1 is not affected by the contraction of s_0, and vice versa. We are in one of the first two cases of the proof of the Critical Pair Lemma (2.7.15). Inspection of these cases shows that the common reducts are found there by contraction of the descendants of the other redex. As a matter of fact, for the first case (disjoint redexes) this is completely obvious. The second case (nested redexes) is slightly more complicated. And we really need left-linearity here: balancing steps would not be descendants.

The proof of the Critical Pair Lemma could be used in a more sophisticated way. In order to trace the descendants, label the head symbol of s_0 with label r (red) and that of s_1 with b (blue). The lifted reduction steps will also be orthogonal, and hence we find a common reduct using the Critical Pair Lemma. In the converging reductions now clearly only coloured redexes are contracted. □

4.2.9. EXERCISE. (i) Give an example showing that the assumption of left-linearity is essential in Proposition 4.2.8.

(ii) Give an example showing that the assumption that there is no overlap is essential in Proposition 4.2.8.

Proposition 4.2.8 restricted to the case of nested redexes would express what after Aczel [1978] is sometimes called the *coherence* of orthogonal rewriting.

We introduce another new terminology. In an orthogonal TRS we have left-linearity and non-ambiguity by definition. So distinct redexes will always give rise to orthogonal steps. In this context, where descendants of redexes will always be redexes again, descendants of redexes are called *residuals*. A redex that is not the residual of a redex is called *created*.

[1] Historically, this proposition goes back to 'Property (D)' in Curry and Feys [1958], here (slightly) paraphrased as follows:

> If R and S are two redexes in X, and the contraction of R followed by contractions of the residuals of S converts X to Y, then a contraction of S followed by contractions of the residuals of R also leads to Y.

4.2.10. PROPOSITION. *Every orthogonal TRS $\mathcal{R}$ has the property* WCR. *Moreover, a common reduct of diverging steps $t \to_{s_0} t_0$ and $t \to_{s_1} t_1$ can be reached from t_0 by contracting the residuals of s_1 and from t_1 by contracting the residuals of s_0.*

PROOF. That $\mathcal{R}$ is WCR follows immediately from the Critical Pair Lemma. For the second part consider diverging steps $t \to_{s_0} t_0$ and $t \to_{s_1} t_1$. Either s_0, s_1 are orthogonal and then we can apply Proposition 4.2.8, or they coincide, and then there are no residuals and $t_0 \equiv t_1$. □

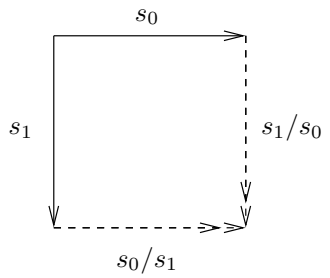


Figure 4.3: WCR for orthogonal TRSs

We introduce the notation s_i/s_j for the residuals of the redex s_i after the reduction step $\to_{s_j}$. Then the situation described in Proposition 4.2.10 is depicted in the diagram in Figure 4.3. It leads to the notion of an *elementary diagram*, to be introduced in Section 4.6. Elementary diagrams will be used there in a confluence proof with a geometric flavour.

4.2.11. REMARK. Consider an orthogonal TRS $\mathcal{R}$. We want to analyse in more detail what can happen in the step $t \to_{s'} t'$ to a redex occurrence s. In order to find the residuals of s we label its head symbol, say by colouring it blue. This way t is transformed to the labelled term t_s. Let the lifted reduction step be $t_s \to_{s'} t''$. Now the following cases can be distinguished, depending on the relative positions of s and s' in t. (Compare the case distinction in the proof of the Critical Pair Lemma (2.7.15).)

(i) *The occurrences of s' and s in t are disjoint.* Then we get back the blue redex s, unaltered, in t''.

(ii) *The occurrences of s and s' coincide.* Then the blue redex has disappeared in t''; t'' does not contain a blue symbol.

(iii) Suppose *s' is a proper subterm of the blue redex s.* By orthogonality s' is contained in one of the arguments of s. Then we get back the blue redex in the same position as the original s, with only some possible change in one of the arguments. Here we need the orthogonality of $\mathcal{R}$ again: Proposition 4.2.10 guarantees that the residual will be redex again.

(iv) Suppose *s is a proper subterm of s'.* Then the blue redex s is n times multiplied for some $n \geq 0$; if $n = 0$, s is erased in t''. The reduct t'' now contains n copies of the blue redex, all of them still blue.

4.2.12. Example. Consider in the TRS

$$\{a \to b,\ b \to c,\ c \to d,\ F(x,y) \to G(y,y),\ G(x,H(y)) \to H(y)\}$$

the reduction step

$$G(F(c,H(a)),H(b)) \ \to\ G(G(H(a),H(a)),H(b))$$

We give the residuals as they arise from the clauses (i)–(iv) in Remark 4.2.11, depending on their position relative to the contracted redex $F(c,H(a))$.

(i) The disjoint redex b has one residual b.
(ii) The contracted redex $F(c,H(a))$ has no residuals.
(iii) The redex $G(F(c,H(a)),H(b))$ has one residual: $G(G(H(a),H(a)),H(b))$.
(iv) The nested redex a is multiplied, it has two residuals a. The other nested redex, c, has no residuals; it is erased.

Then there is one more redex in the end term: $G(H(a),H(a))$. Since it is not the residual of an earlier redex, it is a *created* redex.

The notion of descendant can be generalized to sets of redexes: if S is a set of redex occurrences in t and $t \to_{s'} t'$, then s'' is called a descendant of S if it is a descendant of a redex $s \in S$. We have the following useful characterization of descendants of S, which can be easily verified.

4.2.13. Proposition. *In an orthogonal TRS $\mathcal{R}$ let S be a set of redex occurrences in t and let $t \to_{s'} t'$. Furthermore let t_S be the term that results from t by labelling the head symbols of all redexes in S with the same label ℓ. The step $t \to_{s'} t'$ can be lifted to a step $t_S \to_{s'} t''$ in $\mathcal{R}^{\{\ell\}}$. Then the descendants of S are precisely the redex ocurrences in t' with head symbol labelled in t''.*

Of course this holds also for descendants of sets of function symbols or arbitrary subterm occurrences.

An important consequence is that we can find the descendants of a redex (symbol, subterm) after a sequence of reduction steps without re-initializing the labelling after each single step. We phrase the corollary for symbols.

4.2.14. Corollary. *In an arbitrary TRS $\mathcal{R}$ let t be a term, F a symbol occurrence in t, and let $\rho : t \twoheadrightarrow t'$ be a reduction sequence starting from t. Let t_F be the term that results from t by labelling (only) the symbol occurrence F with ℓ. Then ρ can be lifted to a reduction $\rho_F : t_F \twoheadrightarrow t''$ in the TRS $\mathcal{R}^{\{\ell\}}$. The descendants of the original symbol occurrence F are then precisely the labelled occurrences of F in t''.*

Proof. Induction to the length of ρ, using Proposition 4.2.13. □

We conclude with an exercise. The results will be used in the next section.

4.2.15. Exercise. Consider an orthogonal TRS.

(i) Let t be a term, s a redex occurrence in t, and assume $t \to t'$. Show that the residuals of s in t' will be pairwise disjoint.

(ii) Let t be a term, S a set of pairwise disjoint redex occurrences in t, and assume $t \to t'$. Show that the residuals of S in t' will be pairwise disjoint again.

(iii) Let t be a term, s a redex occurrence in t, and assume $t \twoheadrightarrow t'$. Show that the residuals of s in t' will be pairwise disjoint.

4.3. Confluence via the parallel moves lemma

One of the classical confluence proofs for combinatory logic, which generalizes easily to arbitrary orthogonal TRSs, aims first at the so-called Parallel Moves Lemma (4.3.3). The Parallel Moves Lemma can be seen as arising from a naive attempt to project a single reduction step $t \to_s t'$ over a reduction sequence $t \twoheadrightarrow t''$, as depicted in Figure 4.5. One would like to directly apply Proposition 4.2.10, first to the first step of $t \twoheadrightarrow t''$, then to the second, and so on. Clearly Proposition 4.2.10 cannot be used in this way, however, since the redex s may get multiplied. (See also the chapter on abstract reduction systems on the relation between WCR and CR.)

We look for a notion of reduction that *can* be projected over a single rewrite step, and then, by induction, over an arbitrary reduction sequence $t \twoheadrightarrow t''$. The easiest such notion turns out to be the notion of *parallel reduction*.

4.3.1. Definition. Let a term t contain some disjoint redexes $s_1, s_2, \ldots, s_n$; that is, suppose we have $t \equiv C[s_1, s_2, \ldots, s_n]$ for some context C. Obviously these redexes can be contracted in any order. If their contracta are respectively $s'_1, s'_2, \ldots, s'_n$, in n steps the reduct $t' \equiv C[s'_1, s'_2, \ldots, s'_n]$ can be reached. These n steps together are called a *parallel step*. We write $t \mathrel{-\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow} t'$.

A parallel step is sometimes also called a *parallel move*.

A parallel step *does* nicely project over a single step. This is expressed in the following proposition, here baptized the 'Parallel Moves Proposition' (PMP). From this first the Parallel Moves Lemma and subsequently confluence of orthogonal rewriting will follow with two simple inductions.

We will see several proofs of PMP, using the techniques that are introduced in the following sections. A quite elementary proof can be given as well, in the sense that it uses only low level combinatorial reasoning with the concepts that we have already met. It is somewhat cumbersome to write down all details though, so in our presentation we will leave almost everything to the reader.

4.3.2. Proposition (Parallel Moves Proposition, PMP). *Let $\mathcal{R}$ be an orthogonal TRS, $t \in Ter(\mathcal{R})$. Suppose that $t \mathrel{-\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow} t'$ by contracting disjoint redexes $s_1, \ldots, s_n$ in t, and also $t \to_s t''$ by contracting redex s in t.*

Then a common reduct t''' of t' and t'' can be found as the result of the parallel move $t'' \mathrel{-\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow} t'''$, contracting the residuals of the redexes $s_1, \ldots, s_n$ in

t''. The reduction $t' \twoheadrightarrow t'''$ consists of the contraction of all residuals of s in t' after the reduction $t \twoheadrightarrow t'$. (See Figure 4.4.)

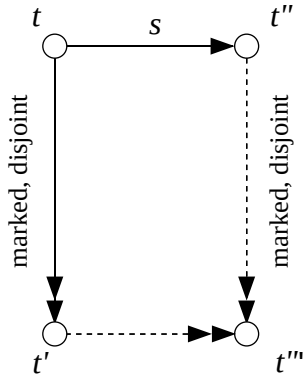


Figure 4.4: Parallel Moves Proposition

PROOF. The proof can be conducted by a tedious (but instructive) case analysis in the style of the proof of the Critical Pair Lemma. We leave it to the reader, who may also choose to skip it here, and instead look at one of the proofs in the following sections. □

An immediate corollary is the Parallel Moves Lemma (PML).

4.3.3. LEMMA (Parallel Moves Lemma). *In an orthogonal TRS let $t \twoheadrightarrow t''$, and let $t \to_s t'$ be a one-step reduction by contraction of redex s. Then a common reduct t''' of t' and t'' can be found by contraction in t'' of all residuals of redex s, which are pairwise disjoint. (See Figure 4.5.)*

PROOF. First note that the step $t \to_s t'$ can also be seen as a parallel move $t \twoheadrightarrow\!\!\!\!\!\!+\; t'$. Then use the Parallel Moves Proposition in an induction on the number of steps in $t \twoheadrightarrow t''$. □

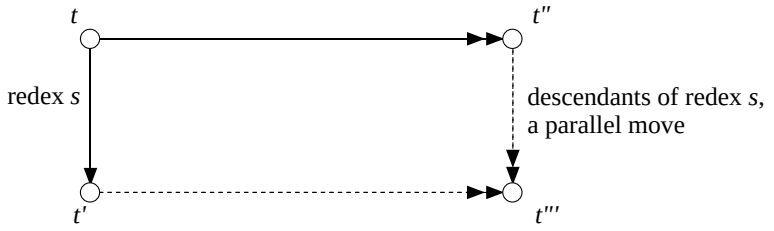


Figure 4.5: Parallel Moves Lemma

By repeated application of the Parallel Moves Lemma one obtains the Church–Rosser Property for orthogonal rewriting.

4.3.4. THEOREM. *Every orthogonal TRS is confluent.*

PROOF. Let $t \twoheadrightarrow t'$ and $t \twoheadrightarrow t''$ and suppose that the first reduction can be specified as

$$t \equiv t_0 \to t_1 \to \cdots t_n \equiv t'$$

Define $s_0 \equiv t''$. By the Parallel Moves Lemma we find a common reduct s_1 of t_1 and s_0, and by repeating this construction we find terms $s_0 \twoheadrightarrow s_1 \twoheadrightarrow \cdots s_n$, such that $t_k \twoheadrightarrow s_k$ for each k, $0 \leq k \leq n$. Then s_n is a common reduct of t' and t''. □

Note that the proof is a special case of Exercise 1.3.6 on ARSs, with both $\to_1$ and $\to_2$ taken as the reduction relation $\to$ of the orthogonal TRS.

4.4. Underlinable reductions

Our second confluence proof will be more systematic and sophisticated than that of the previous section. We use a labelling technique again, called *underlining*, but now in such a way that all explicit mention of residuals can in principle be avoided. Underlined reduction differs from labelled reduction as we used it up till now, in that it not only *records* reduction, but also *restricts* it: in the underlined TRS, to be defined below, only redexes with underlined head symbol may be contracted.

4.4.1. DEFINITION (underlined reduction). Let $\mathcal{R} = (\Sigma, R)$ be a TRS. We define the *underlined* TRS $\underline{\mathcal{R}} = (\underline{\Sigma}, \underline{R})$.

(i) The underlined signature $\underline{\Sigma}$ is defined as the union

$$\underline{\Sigma} = \Sigma \cup \{\underline{f} \mid f \in \Sigma\}$$

The arity of a function symbol $\underline{f}$ is the same as that of f. The terms over $\underline{\Sigma}$ are the *underlined* terms.

(ii) The *underlined* version $\underline{\rho}$ of a reduction rule ρ is obtained by underlining (only) the head symbol of the left-hand side of ρ.

More precisely, for $\rho : l \to r$, with $l \equiv f(t_1, \ldots, t_n)$, we have the underlined version $\underline{\rho} : \underline{f}(t_1, \ldots, t_n) \to r$.

(iii) The set $\underline{R}$ of underlined rules is defined as

$$\underline{R} = \{\underline{\rho} \mid \rho \in R\}$$

Note that the underlined TRS $\underline{\mathcal{R}}$ can be considered as a labelled TRS with a single label u, written as underlining, and with a restricted set of reduction rules. We have $\underline{\Sigma} = \Sigma^{\{u\}}$ and $\underline{R} \subset R^{\{u\}}$. Note especially that the original rules of R are not included in $\underline{R}$. Just as we could use different labels simultaneously, we can also use different kinds of underlining, say distinguished by their colour.

4.4.2. EXAMPLE. The underlined version of combinatory logic, $\underline{\mathrm{CL}}$, contains the following underlined reduction rules:

$$\begin{array}{lcl} \underline{Ap}(Ap(Ap(S,x),y),z) & \to & Ap(Ap(x,z),Ap(y,z)) \\ \underline{Ap}(Ap(K,x),y) & \to & x \\ \underline{Ap}(I,x) & \to & x \end{array}$$

Since all underlined reduction is also labelled reduction, we know that if $t \twoheadrightarrow_{\underline{\mathcal{R}}} t'$, then the underlined symbols in t' are descendants of the underlined symbols in t (by Proposition 4.2.13 and Corollary 4.2.14). This leads to the following proposition.

4.4.3. PROPOSITION. *Let $\mathcal{R}$ be an orthogonal TRS. Let $t \twoheadrightarrow_{\underline{\mathcal{R}}} t'$, where t is a term in which only head symbols of redexes are underlined. Then also in t' all underlined symbols will be head symbols of redexes.*

PROOF. By the above observation the subterms with underlined head symbol in t' are residuals of the redexes with underlined head symbol in t. We can then apply Proposition 4.2.8 (the key property of orthogonal rewriting). □

4.4.4. PROPOSITION. *Let $\mathcal{R}$ be an orthogonal TRS. Then $\underline{\mathcal{R}}$ is also orthogonal.*

PROOF. Both left-linearity and non-ambiguity carry over immediately from $\mathcal{R}$ to $\underline{\mathcal{R}}$. For example, if we had overlap between two underlined reduction rules, this same overlap would exist between the un-underlined originals of these rules in $\mathcal{R}$. □

4.4.5. PROPOSITION. *Underlined reduction is strongly normalizing. That is, for any TRS $\mathcal{R}$ the underlined version $\underline{\mathcal{R}}$ is SN.*

PROOF. Define a complexity measure $\nu(t)$ for $t \in Ter(\underline{\mathcal{R}})$ as follows:

$$\begin{array}{lcl} \nu(x) & = & 0 \\ \nu(c) & = & 0 \\ \nu(\underline{c}) & = & 1 \\ \nu(f(t_1,\ldots,t_n)) & = & max\{\nu(t_1),\ldots,\nu(t_n)\} \\ \nu(\underline{f}(t_1,\ldots,t_n)) & = & 1 + max\{\nu(t_1),\ldots,\nu(t_n)\} \end{array}$$

Now consider in a term $t \in Ter(\underline{R})$ all subterm occurrences of which the head symbol is underlined, call these the underlined subterms. Of these underlined subterms s, we collect their measure $\nu(s)$ in a multiset called $\|t\|$, the *norm* of t. So $\|t\| = [\,\nu(\underline{f}(t_1,\ldots,t_n)) \mid \underline{f}(t_1,\ldots,t_n) \leq t\,]$. Here $[\ldots]$ denotes a multiset of natural numbers. Example: Consider in the TRS $\mathcal{N}^{AM}$ for addition and multiplication in Table 2.3 the term

$$t = A(M(\underline{M}(0,\underline{0}),0),\underline{M}(\underline{0},S(\underline{A}(0,\underline{A}(\underline{0},\underline{A}(0,\underline{0})))))))$$

Then

$$\|t\| = [2, 1, 5, 1, 4, 3, 1, 2, 1]$$

Now it is not hard to prove that $s \to t$ implies that $\|s\| >_{\#} \|t\|$, where $>_{\#}$ is the well-founded multiset order on multisets of natural numbers explained in the Appendix, Section A.2. □

4.4.6. Remark. A shorter proof of termination of underlined reduction can be given using the method of recursive path orders, as found in Chapter 6 of Terese [2003]. Just employ an ordering of symbols such that $\underline{F} \succ F$, for all symbols $F \in \Sigma$.

4.4.7. Proposition. *Let $\mathcal{R}$ be orthogonal. Then $\underline{\mathcal{R}}$ is confluent.*

Proof. Since $\mathcal{R}$ is orthogonal, so is $\underline{\mathcal{R}}$. Then it follows by the Critical Pair Lemma (2.7.15) that $\underline{\mathcal{R}}$ is WCR. For, if there are no critical pairs, then all critical pairs are trivially convergent. Since $\underline{\mathcal{R}}$ is also SN (Proposition 4.4.5), Newman's Lemma (1.2.1) then yields CR. □

4.4.8. Exercise. Give an example of a TRS $\mathcal{R}$ which is not orthogonal such that its underlined version $\underline{\mathcal{R}}$ is not confluent.

4.4.9. Remark. Let $t \in \mathit{Ter}(\underline{\mathcal{R}})$ be such that only some head symbols of *redexes* are underlined. Reduce t to its unique $\underline{\mathcal{R}}$-normal form t^*. Then the term t^* contains no underlining at all. This follows from the fact that in t^*, being a reduct of t, only head symbols of redexes can be underlined, combined with the fact that t^*, being a normal form, contains no redexes.

4.4.10. Definition. Let $s, t \in \mathit{Ter}(\underline{\mathcal{R}})$. We define $t \geq s$, if t can be obtained from s by adding some underlinings, i.e. by replacing some symbols F by $\underline{F}$, but only for Fs that are in $|t|$ the head symbols of redexes.

The following proposition says that adding more underlining in the sense of $\geq$ does not hinder reduction.

4.4.11. Proposition (lifting). *Suppose $t' \geq t \twoheadrightarrow_{\underline{\mathcal{R}}} s$. Then there is a term s' such that $t' \twoheadrightarrow_{\underline{\mathcal{R}}} s' \geq s$. (See Figure 4.6.)*

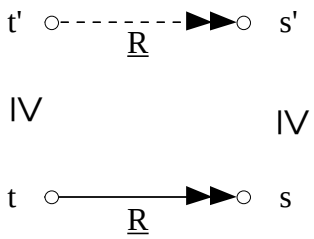


Figure 4.6: Adding underlining

PROOF. An easy induction on the length of $t \twoheadrightarrow_{\underline{\mathcal{R}}} s$. For the basis step it is essential that no underlinings are added to other symbols in the pattern of a redex than the head symbol. □

4.4.12. REMARK. Note that left-linearity is necessary in Proposition 4.4.11. For let $R = \{D(x,x) \to 0,\ 0 \to 1\}$. Then $\underline{D}(\underline{0},0) \geq \underline{D}(0,0) \to_{\underline{\mathcal{R}}} 0$, but there is no term $s' \geq 0$ such that $\underline{D}(\underline{0},0) \twoheadrightarrow_{\underline{\mathcal{R}}} s'$.

4.4.13. DEFINITION (underlinable reduction). A reduction $s \twoheadrightarrow t$ in $\mathcal{R}$ is called *underlinable*, notation $s \underline{\twoheadrightarrow} t$, if there is a reduction $s' \twoheadrightarrow t'$ in $\underline{\mathcal{R}}$ such that $s \geq s'$ and $t \geq t'$. One also says that the reduction $s \underline{\twoheadrightarrow} t$ in $\mathcal{R}$ is *lifted* to $s' \twoheadrightarrow t'$ in $\underline{\mathcal{R}}$.

4.4.14. EXAMPLE. Let $\mathcal{R}$ be the TRS $\mathcal{N}^{AM}$ for addition A and multiplication M of Table 2.3.

(i) The reduction step $M(0,S(0)) \to_{\mathcal{R}} A(M(0,0),0)$ is underlinable:

$$\underline{M}(0,S(0)) \to_{\underline{\mathcal{R}}} A(M(0,0),0)$$

(ii) The reduction sequence $A(A(0,0),0) \to_{\mathcal{R}} A(0,0) \to_{\mathcal{R}} 0$ is underlinable:

$$\underline{A}(\underline{A}(0,0),0) \to_{\underline{\mathcal{R}}} \underline{A}(0,0) \to_{\mathcal{R}} 0$$

(iii) The reduction sequence $M(0,S(0)) \to_{\mathcal{R}} A(M(0,0),0) \to_{\mathcal{R}} A(0,0)$ is not underlinable.

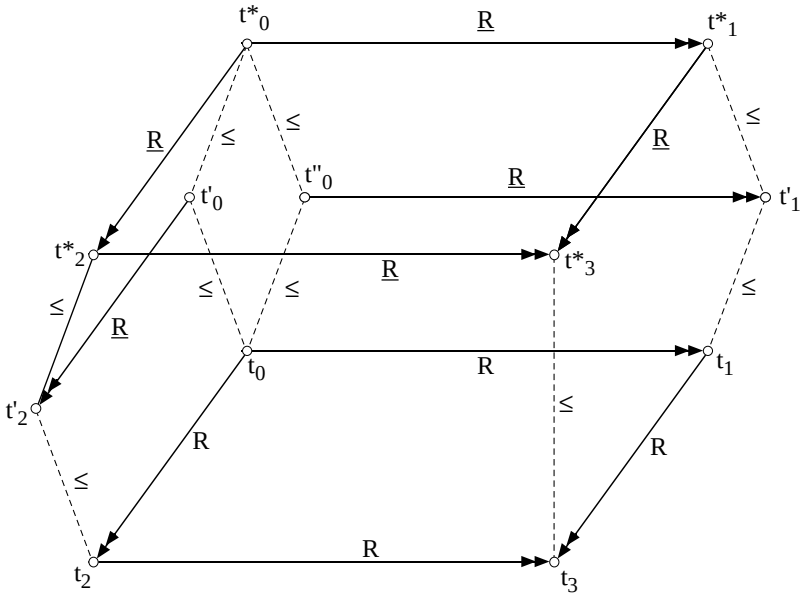


Figure 4.7: Diamond property for underlinable reductions

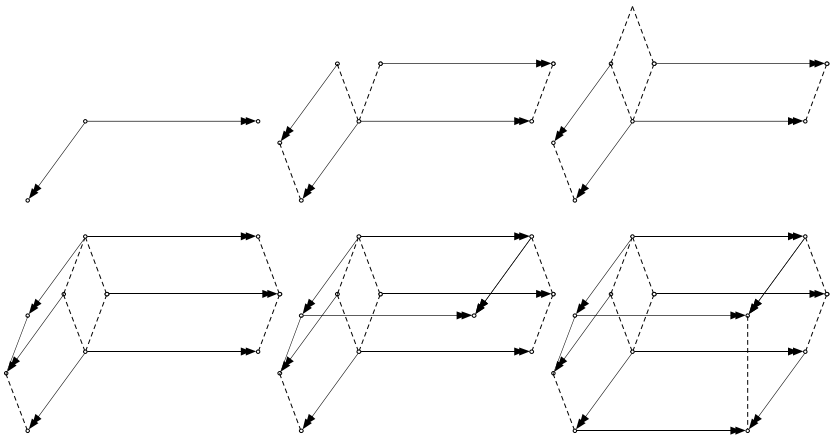


Figure 4.8: Construction of diagram in Figure 4.7

4.4.15. PROPOSITION. *Let $\mathcal{R}$ be an orthogonal TRS. Suppose $t_0 \underline{\twoheadrightarrow} t_1$ and $t_0 \underline{\twoheadrightarrow} t_2$. Then there is a common reduct t_3 such that $t_1 \underline{\twoheadrightarrow} t_3$ and $t_2 \underline{\twoheadrightarrow} t_3$.*

PROOF. The proof is illustrated in Figures 4.7 and 4.8. As $t_0 \twoheadrightarrow_{\mathcal{R}} t_1$ and $t_0 \twoheadrightarrow_{\mathcal{R}} t_2$ are underlinable reductions, there are terms t_0', t_1', t_0'', t_2' such that $t_0 \leq t_0'$, $t_1 \leq t_1'$, $t_0 \leq t_0''$, $t_2 \leq t_2'$, $t_0' \twoheadrightarrow_{\underline{\mathcal{R}}} t_1'$, and $t_0'' \twoheadrightarrow_{\underline{\mathcal{R}}} t_2'$. Taking the union of the underlinings in t_0' and t_0'' we find a term t_0^* such that $t_0' \leq t_0^*$ and $t_0'' \leq t_0^*$. By Proposition 4.4.11 there are terms t_1^* and t_2^* such that $t_1 \leq t_1^*$, $t_2' \leq t_2^*$, $t_0^* \twoheadrightarrow_{\underline{\mathcal{R}}} t_1^*$, $t_0^* \twoheadrightarrow_{\underline{\mathcal{R}}} t_2^*$.

Because $\underline{\mathcal{R}}$ is confluent, there is a term t_3^* in $\underline{\mathcal{R}}$ such that $t_1^* \twoheadrightarrow_{\underline{\mathcal{R}}} t_3^*$ and $t_2^* \twoheadrightarrow_{\underline{\mathcal{R}}} t_3^*$. Clearly these reductions are again underlinable. □

Thus we arrive, via the diamond property for $\underline{\twoheadrightarrow}$, at the second proof of CR (cf. Theorem 4.3.4).

4.4.16. THEOREM. *Every orthogonal TRS is confluent.*

PROOF. Immediate, using Propositions 1.1.11 and 4.4.15, noting that a single reduction step is always underlinable (hence $\to \subseteq \underline{\twoheadrightarrow}$) and that obviously $\underline{\twoheadrightarrow} \subseteq \twoheadrightarrow$. See also Figure 4.9. □

4.5. Complete developments

Our third variation of the confluence proof will highlight a notion that is classic in the traditions of λ-calculus and combinatory logic, namely that of *complete developments*. This notion also applies to orthogonal TRSs.

What is a development? We will give a more formal definition, but one can say it in words. A *development* from t is a reduction in which first a redex in t is contracted, and after that only residuals of redexes in t. A development from t is *complete* if no residuals of redexes in t are left. The

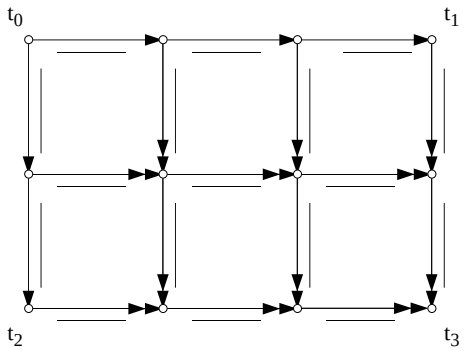


Figure 4.9: Underlinable reductions yield confluence

notion of development can be relativized to a set S of redexes (not necessarily all) in t: first a redex from the set S, afterwards only residuals of S.

4.5.1. DEFINITION. Let $\mathcal{R}$ be an orthogonal TRS, t a term in $\mathcal{R}$, and s a redex occurrence in t and S a set of redex occurrences in t.

(i) By t_s we denote the underlined term that results from t by underlining only the head symbol of s. We also write $t \underline{\leq}_s t_s$.

(ii) By t_S we denote the underlined term that results from t by underlining the head symbol of all redex occurrences $s \in S$. We also write $t \underline{\leq}_S t_S$.

4.5.2. DEFINITION. Let $\mathcal{R}$ be an orthogonal TRS, t a term in $\mathcal{R}$, and S a set of redex occurrences in t.

(i) A *development* of S in t is a reduction $t \twoheadrightarrow t'$ in $\mathcal{R}$ that can be lifted to a reduction $t_S \twoheadrightarrow t''$ in $\underline{\mathcal{R}}$, such that $t' \underline{\leq} t''$.

(ii) A development as in (i) is a *complete development* of S in t if $t' \equiv t''$. (So in t'' no underlined symbols remain.) In this case we write $t \twoheadrightarrow_S t'$.

(iii) A *(complete) development of* t is a (complete) development of S in t, where S is the set of all redex occurrences in t.

One can now verify that this definition matches the informal description of (complete) developments given above. Note that also a single reduction step can count as a complete development, namely with respect to the singleton set consisting of just the contracted redex itself. There is also a direct correspondence with the notion of underlinable reduction of the previous section.

4.5.3. EXERCISE. Recall the notion of underlinable reduction, $\underline{\twoheadrightarrow}$, from Definition 4.4.13. Verify that an underlinable reduction from t is, in our new terminology, the same as a development of t, i.e., an arbitrary reduction from t in which no created redexes are contracted.

The following is another specimen of a classic result in the theory of term rewriting, stemming from the λ-calculus and CL tradition.

4.5.4. THEOREM (Finite Developments Theorem). *All developments are finite.*

PROOF. This follows immediately by SN for underlined reduction, Proposition 4.4.5. □

The property expressed in the Finite Developments Theorem is known as FD. Using Proposition 4.4.7, stating that underlined reduction in an orthogonal TRS is CR, it can be strengthened to what is known as FD!: given a term t with a set of redexes S, all complete developments of S from t end in the same term t'. This term can be reached by just starting a development of S, prolonging it in an arbitrary fashion until no residuals of S remain.

4.5.5. EXERCISE (FD!). Let $\mathcal{R}$ be an orthogonal TRS.

(i) Show that given a term t in $\mathcal{R}$ and a set S of redex occurrences in t, there is exactly one term t' such that $t \twoheadrightarrow_S t'$.

(ii) Show that given a term t in $\mathcal{R}$, there is exactly one term t' such that all complete developments of t end in t'. (This *term* t' is also sometimes called the *complete development* of t.)

The property FD! can be further strengthened, to account for residuals of redexes that are not in the set S being developed.

4.5.6. PROPOSITION (FD$^+$). *Let $\mathcal{R}$ be an orthogonal TRS, and $t \in Ter(\mathcal{R})$. Let S and T be sets of redexes in t and let $t \twoheadrightarrow_S t'$. Then the set of residuals of T in t' is unique, that is, independent of the order in which redexes in S and their residuals are contracted in a complete development from t.*

PROOF. In the underlined term t_S the redexes in T can be marked, say by colouring their head symbols blue. Call the resulting term ${t_S}^T$. The developments of S correspond exactly to the underlined reductions from t_S, and these can be lifted further to coloured underlined reductions from ${t_S}^T$. Since both underlining and colouring (or labelling) preserve the orthogonality of $\mathcal{R}$, the normal form of ${t_S}^T$ must be unique. This normal form, in which no underlining is left, is a coloured version of t'. The redexes with a blue head symbol are the residuals of T. □

4.5.7. NOTATION. Let S and T be sets of redexes in a term t in an orthogonal TRS. By T/S we denote the set of residuals of T after a complete development of S from t.

We now head for a confluence proof of orthogonal rewriting again, But we will this time be able to get some additional information from it.

4.5.8. PROPOSITION. *Let $\mathcal{R}$ be an orthogonal TRS, and $t \in Ter(\mathcal{R})$. Let S and T be sets of redexes in t, and let t' and t'' be terms such that $t \twoheadrightarrow_S t'$ and $t \twoheadrightarrow_T t''$. Then there exists a common reduct t''' of t' and t'', such that $t' \twoheadrightarrow_{T/S} t'''$ and $t'' \twoheadrightarrow_{S/T} t'''$. (See Figure 4.10(a).)*

PROOF. Consider the set $S \cup T$. The common reduct t''' can be found as the unique $\underline{\mathcal{R}}$-normal form of $t_{S\cup T}$. By lifting the reduction $t \twoheadrightarrow_S t'$ we have $t_{S\cup T} \twoheadrightarrow_{\underline{\mathcal{R}}} t'_{(S\cup T)/S}$. Since no residuals of S remain in t', we have in fact $t_{S\cup T} \twoheadrightarrow_{\underline{\mathcal{R}}} t'_{T/S}$. Then t''' is reached by reducing $t'_{T/S}$ to normal form in $\underline{\mathcal{R}}$. That is $t' \twoheadrightarrow_{T/S} t'''$ by Definition 4.5.2(ii). Likewise $t'' \twoheadrightarrow_{S/T} t'''$.

One can think of the redexes in S as red redexes, and those in T as blue, as in Figure 4.10(b). (The redexes that belong to both S and T get two colours.) Then the lifted reduction contracts first only red redexes, until in $t'_{T/S}$ no red ones are left, and then, for lack of red redexes only blue ones, until t''' is reached. These blue redexes are all residuals of the blue redexes belonging to the set T/S in t'. □

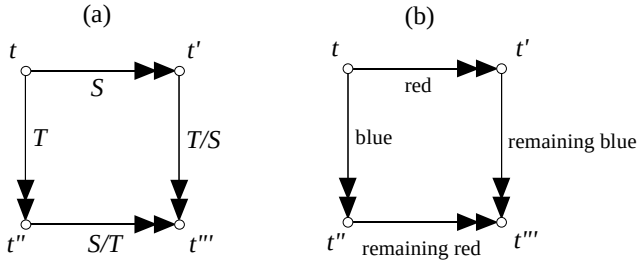


Figure 4.10: Diamond property of complete developments

Note that this proposition can be seen as a generalization of the Parallel Moves Proposition (4.3.2).

4.5.9. COROLLARY. *The Parallel Moves Proposition.*

PROOF. We get the PMP if we apply Proposition 4.5.8 with $T = \{s_0, \ldots, s_n\}$ a set of disjoint redexes in t and $S = \{s\}$ a single redex. □

A useful particular case of the above proposition is that where the set T of redexes is a subset of S. Then one of the converging complete developments is in fact the *empty reduction* (or the complete development of the empty set of redexes), see Figure 4.11. So in fact $t_3 \equiv t_1$. This might be called the *triangle property* for the reason depicted in Figure 4.11(b). (But see Remark 4.5.13 below.) We have proved the following.

4.5.10. PROPOSITION. *Let $t_0 \twoheadrightarrow_S t_1$ be a complete development of a set S of redexes in t_0, and let $t_0 \twoheadrightarrow_T t_2$ be a complete development of a set T of redexes in t_0, such that $T \subseteq S$. Then there is a complete development from t_2 to t_1 (with respect to S/T).*

PROOF. We have the situation of Proposition 4.5.8 with $T/S = \varnothing$. So $t_1 \equiv t_3$ and hence $t_2 \twoheadrightarrow_{S/T} t_1$. □

To be able to finish the confluence proof we define what is often called *simultaneous rewriting*. Here we call it *multi-step* reduction.

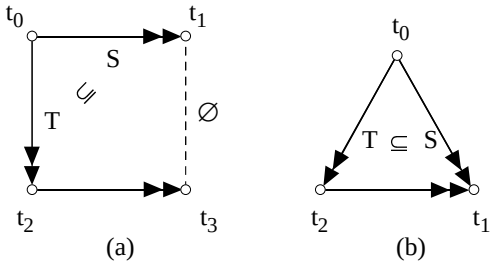


Figure 4.11: Triangle property of complete developments

4.5.11. Definition. If $t \twoheadrightarrow_S t'$ for some S, we write $t \multimap t'$. This will be called a *multi-step*.

4.5.12. Proposition. *Let $\mathcal{R}$ be an orthogonal TRS.*

(i) *The relation $\multimap$ satisfies the triangle property (*TP*).*

(ii) *The relation $\multimap$ satisfies the diamond property (*DP*).*

Proof. (i) Let S be the set of all redexes in t and let t^* be the unique term such that $t \twoheadrightarrow_S t^*$. Assume $t \multimap t'$. We show that $t' \multimap t^*$. By the definition of $\multimap$ we have $t \twoheadrightarrow_T t'$ for some set of redexes T. Since $T \subseteq S$ it then follows by Proposition 4.5.10 that $t' \twoheadrightarrow_{S/T} t^*$. Hence $t' \multimap t^*$.

(ii) Immediate by (i), as in general TP implies DP. □

4.5.13. Remark. Proposition 4.5.10 still refers to sets of redexes (and their residuals). This information is abstracted away in the triangle property of Proposition 4.5.12. The additional structure of Proposition 4.5.10 is reflected in the name *Prism* Theorem which was used for a similar property in the λ-calculus in Huet [1994].

4.5.14. Corollary. *Confluence of orthogonal rewriting.*

Proof. Propositions 1.1.11 and 4.5.12 can be applied, since we obviously have $\rightarrow \subseteq \multimap \subseteq \twoheadrightarrow$. □

4.6. Completion of reduction diagrams

The preceding three confluence proofs are all constructive in the sense that they can in principle be used to actually find a common reduct for a concrete case of diverging reductions. However, the algorithm is somewhat implicit in these proofs. A very explicit algorithm is given by the next variation of the confluence proof. In this proof we will adopt a geometric view. The informal representation of diverging and converging reductions in diagrams, which up till now was used only for the purpose of illustration, will be taken seriously, and we will actually reason in terms of such geometric representations. In fact, the theorem that will be proved strengthens the confluence theorem, by

showing that confluence can be obtained in a canonical way, namely by tiling with *elementary diagrams*.

4.6.1. Elementary diagrams

Elementary diagrams (e.d.'s) are the simplest possible reduction diagrams, obtained by completing two co-initial diverging reduction steps with reductions towards a common reduct in what is clearly the most natural way, namely by contracting the residuals of the vertical redex after contraction of the horizontal redex on the one hand, and contracting the residuals of the horizontal redex after contraction of the vertical redex on the other hand. Because of Proposition 4.2.10 we can be sure that this will always work. Recall that the residuals of a redex are always disjoint, and can hence be contracted independently and in any order. As before, s_i/s_j denotes the residuals of the redex s_i after the step $\to_{s_j}$.

4.6.1. Definition. In an orthogonal TRS consider a term t with redex occurrences s_0, s_1 and let $t \to_{s_0} t'$ and $t \to_{s_1} t''$. The corresponding *elementary diagram* (e.d.) is a rectangular configuration with *initial reduction steps* $t \to_{s_0} t'$ and $t \to_{s_1} t''$, as upper and left sides, respectively. The lower and right sides consist of subsequent contractions of residual redexes, the lower side of the redexes in s_0/s_1 and the right side of those in s_1/s_0. These reductions end in the same term t''', which is the lower-right corner of the e.d.

4.6.2. Example. An example is given in Figure 4.12(a). The CL-term $Sab(Ic)$ contains two redexes, the term itself and the redex Ic. The small redex Ic is contracted in the horizontal step, the big one in the vertical step. The converging vertical reduction contracts $Sabc$, the residual of $Sab(Ic)$ after contraction of Ic; the converging horizontal reduction contracts the two residuals of Ic after contraction of $Sab(Ic)$. The faint arrows in the figure suggest how the residuals 'propagate' to the other sides of the elementary diagram. For the sake of definiteness, we will stipulate that the two residuals in the horizontal reduction are contracted in left-to-right order; later we will see that the other possibilities yield essentially equivalent results. In the example of Figure 4.12(a), the upper redex Ic had two residuals, contracted in the lower reduction; in general this multiplicative effect may take any value n, also $n = 0$, as is the case in the example of Figure 4.12(b). In the case $n = 0$, we say that the redex leaving no residuals is *erased*.

Note that in the case of nested redexes, contraction of the outer redex can multiply the other one, but also erase it. Therefore the lower or right side of an e.d. may consist of several steps, or one, or none at all. In case one side consists of several steps, we call the e.d. *splitting*. The order of these steps has not been fixed. However, since the residuals of the initial redex are disjoint, it does not make much difference which order we choose, and we will neglect the issue. It is essential, however, that the steps are performed sequentially, in some order, and not in parallel.

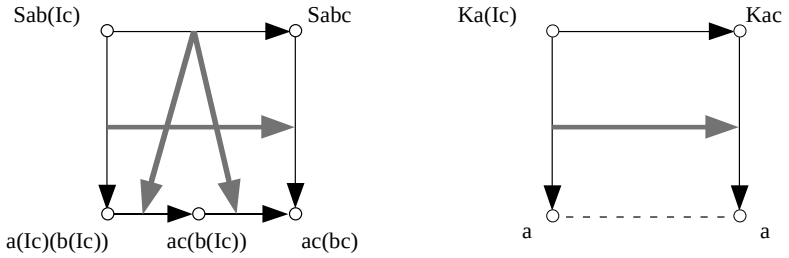


Figure 4.12: Elementary diagrams in CL

When there are no residuals to contract, the corresponding side contains no reduction steps at all. We call this side an *empty step* and draw a dashed line, just to keep the diagram rectangular. This is essential for the process of tiling that we are going to define, which will use the e.d.'s as the building stones. Non-empty steps are then called *proper*. Note that the endpoints of a dashed line are always identical terms.

Having introduced empty steps, we also have the situation where an empty step meets a proper step; and even the case that two empty steps meet. All this gives rise to e.d.'s of the types displayed in Figure 4.13. The three lower types are called *improper* e.d.'s.

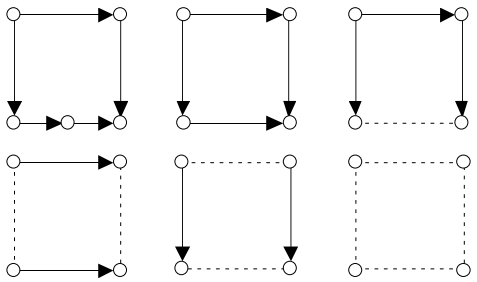


Figure 4.13: Types of elementary diagrams

4.6.2. Reduction diagrams

Let t_0 be a term in an orthogonal TRS with reductions $t_0 \twoheadrightarrow t_1$ and $t_0 \twoheadrightarrow t_2$. We try to find a common reduct of t_0, t_1 by a *tiling* procedure in an attempt to construct a *completed reduction diagram.*

The construction of such a diagram starts by setting out the initial reductions, one horizontally, one vertically, both starting at the same point, representing t_0. This configuration is depicted in Figure 4.14, it is called an *initial conversion*.

The construction then proceeds by subsequently adjoining e.d.'s at *open corners.* Here, by an *open corner* we mean two diverging steps, one to the

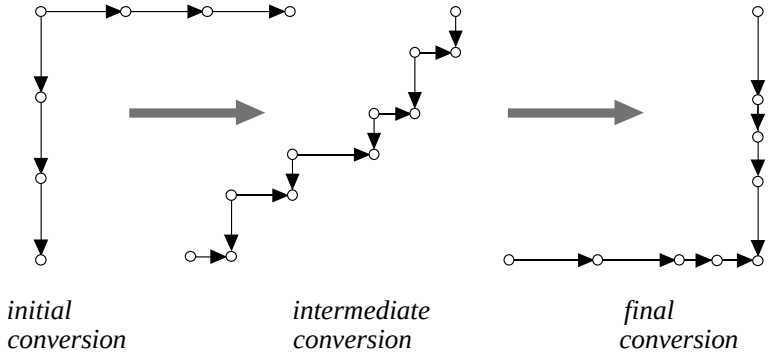


Figure 4.14: Conversions

right and one downward, without converging sides.

Adjunction of an e.d. will always be possible: if the diverging steps are *proper* steps, then by orthogonality; if one or both are *empty* we can use one of the types of improper e.d.'s.

Each stage in the tiling process just described is called a *reduction diagram*. A reduction diagram is *completed* if it has no open corners. With a completed diagram the tiling process terminates. It is easy to see that a completed reduction diagram with initial reductions $t_0 \twoheadrightarrow t_1$, $t_0 \twoheadrightarrow t_2$ has convergent reductions for t_1 and t_2 at the bottom and right. Figure 4.15 shows a completed reduction diagram.

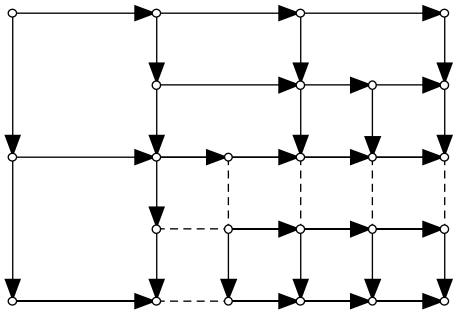


Figure 4.15: Completed reduction diagram

There is another way of looking at the same process. With each adjunction of an e.d. we transform the conversion, starting with an initial conversion, passing through subsequent stages of *intermediate conversions*, until a *final conversion* is reached. This view is depicted in Figure 4.14.

A priori, this attempt at constructing a completed reduction diagram (or a final conversion) could fail, and end up in an infinite regress leading to Escher-like figures as depicted in Figure 4.16. However, in the present situation of orthogonal TRSs, the attempt will always succeed. This is what we will prove in this subsection. If we succeed we will in fact have shown more than just CR. The tiling is done with e.d.'s, with on the converging sides contractions of

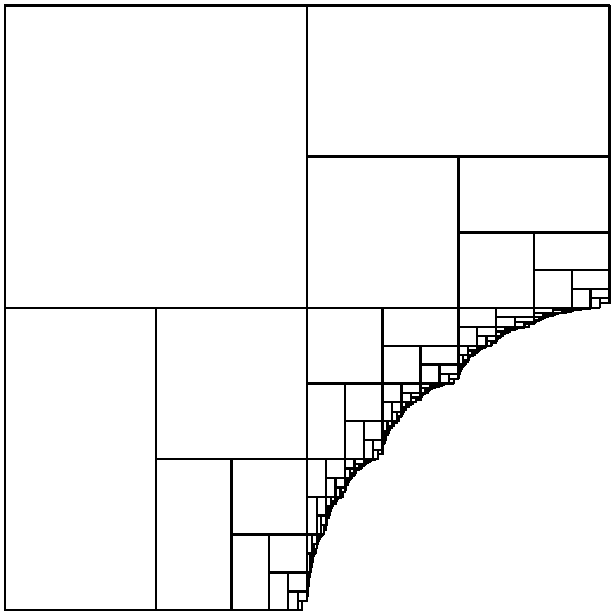


Figure 4.16: Fractal- or Escher-like figure

residuals of the redexes corresponding to the initial steps. As a consequence in the completed reduction diagram we will have a pattern of propagation of residuals as depicted in Figure 4.17. All redexes contracted in horizontal steps will be residuals of redexes that were contracted in the initial horizontal reduction. For the vertical steps we have the same.

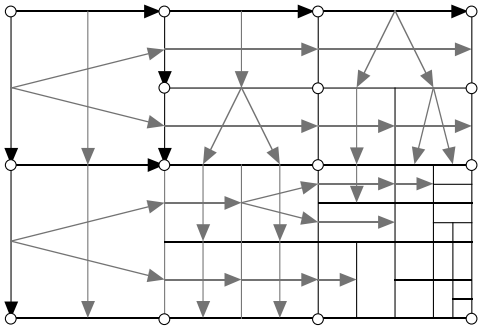


Figure 4.17: Propagation of residuals in a completed reduction diagram

We start by showing that the tiling procedure terminates in the simpler situation where the initial reductions are complete developments.

4.6.3. Proposition. *Let in an orthogonal TRS an initial conversion be given, consisting of two diverging complete developments. Then the tiling procedure terminates in a completed reduction diagram, where the converging sides are again complete developments.*

Proof. Consider an initial conversion consisting of a horizontal reduction

$t_0 \twoheadrightarrow t_1$ which is a complete development of red-underlined redexes, and a vertical reduction $t_0 \twoheadrightarrow t_2$ which is a complete development of blue-underlined redexes. See Figure 4.14: the tiling process of adding e.d.'s will lead to successive intermediate conversions, with the aim of reaching a final conversion. Clearly, an invariant of the tiling process is that the proper horizontal steps are contracting redexes with red underlining, and the proper vertical steps are contracting redexes with blue underlining.

Now we will show that the successive conversions are in fact decreasing with respect to some well-founded ordering. First note that in an orthogonal TRS each term has only finitely many one-step reducts. (This will hold even if infinitely many rewrite rules are allowed, since a finite term can contain only finitely many non-overlapping redex patterns.)

We have proved that red–blue-underlined reductions are terminating. Combining this with our observation of finitely many one-step reducts for each term, we have, by König's Lemma (A.1.25), that a red–blue-underlined term has only finitely many reducts with respect to red–blue-underlined reduction. The finiteness of this number of reducts gives us a notion of *weight* of an underlined term, by means of which we define a *norm* on intermediate conversions Γ.

(i) Let t be a term with red–blue underlining. Then the *weight* $w(t)$ of t is the number of reducts of t with respect to red–blue reduction.

(ii) Let $\Gamma = s_1 \leftrightarrow s_2 \leftrightarrow \cdots \leftrightarrow s_n$ be an intermediate conversion as in Figure 4.14; each $\leftrightarrow$ is either $\rightarrow$ or $\leftarrow$. Then the *norm* of Γ, notation $\|\Gamma\|$, is the multiset of the weights of the terms involved:

$$\|\Gamma\| = [\, w(s_1), \ldots, w(s_n) \,].$$

Here $[\,,\,]$ denote multiset brackets.

Then the weight of a term has the property that it decreases during (red–blue) reduction: if $t \rightarrow t'$ then $w(t) > w(t')$.

Consider now the effect of adjoining a proper e.d. to Γ. Let the resulting conversion be Γ'. So we have

$$\begin{aligned} \Gamma &= \cdots s_2 \leftarrow s_0 \rightarrow s_1 \cdots \\ \Gamma' &= \cdots s_2 \rightarrow s_4 \leftarrow s_3 \leftarrow s_1 \cdots \end{aligned}$$

For the multisets $\|\Gamma\|$, $\|\Gamma'\|$ this means that $w(s_0)$ is replaced by $w(s_4)$, $w(s_3)$; and since $w(s_0) > w(s_4), w(s_3)$, we have $\|\Gamma\| >_{\#} \|\Gamma'\|$, where $>_{\#}$ is the multiset ordering.

If we adjoin an improper e.d., then there is no gain: $\|\Gamma\| = \|\Gamma'\|$. But it is easy to see that such an improper e.d. can be only finitely many times adjoined, after which, if a final conversion is not reached yet, another decrease in the multiset must follow. The crucial point is here that an improper e.d. is *unsplitting*: the lower and right side consist each of one step (at least one

of which must be an empty step). For reasons of simple geometry an infinite sequence of adjunctions would have to involve indefinitely shrinking e.d.'s. But this is impossible if there is no splitting.

So, by well-foundedness of the ordering on multisets of natural numbers, the tiling procedure must terminate. Then we have reached a final conversion, that is, two converging reductions; by the invariant mentioned above, the vertical one consists of blue-underlined steps, and the horizontal one consists of red-underlined redex contractions. In the common reduct where they meet, there can be no underlining present, so the converging reductions are indeed complete developments. □

4.6.4. THEOREM. *Let in an orthogonal TRS a conversion* Γ_0 *be given. Then the tiling procedure, starting from* Γ_0*, terminates in a completed reduction diagram.*

PROOF. Consider Figure 4.18, where the starting conversion is Γ_0. We divide the 'working space', where the tiling procedure is taking place, into 'main squares' as in the figure, determined by the steps in the conversion Γ_0, simply by drawing horizontal and vertical lines from the points (terms) on Γ_0. Note that each side of a main square, when in the tiling process it gets filled in with actual reduction steps, can be seen as a development. (If it is horizontal it consists of the contraction of residuals of the redex contracted in the original step in Γ_0 above it.) When it is completely filled it will be a complete development. Therefore, by Proposition 4.6.3 each main square can contain only finitely many e.d.'s. Since we have a finite number of main squares, the tiling procedure must then terminate. □

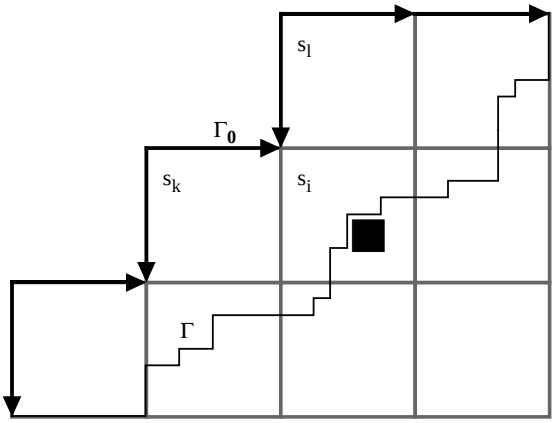


Figure 4.18: Adding an e.d. to intermediate conversion

4.6.3. Permutation equivalence

With Theorem 4.6.4 at our disposal, we now define the very useful notion of the *projection* of a reduction ρ over a reduction σ.

4.6.5. Definition. Let ρ, σ be finite reductions in an orthogonal TRS. Let $\mathcal{D}$ be the completed reduction diagram obtained by completing (as in Theorem 4.6.4) the horizontal reduction ρ against the vertical reduction σ; see Figure 4.19. Then the reduction at the lower side of $\mathcal{D}$ is the *projection* of ρ over σ, notation ρ/σ.

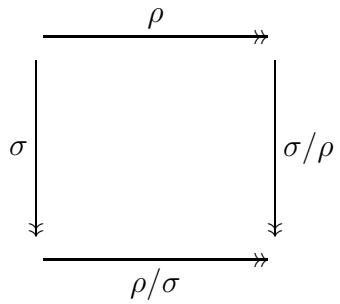


Figure 4.19: Projections

4.6.6. Remark. (i) Note that the projection ρ/σ also makes sense if ρ is an infinite reduction, but σ should be finite.

(ii) Since the construction of a reduction diagram possibly involves empty steps, the projection may even consist entirely of empty steps – we call it an *empty reduction*. Both empty steps and empty reductions are here denoted by $\varnothing$.

(iii) The equations generated by the reduction relation of Excercise 2.7.17 hold for the projection operator /. As a matter fact, they were designed as such.

The projection operation yields a natural equivalence on reductions. It turns out to be equivalent to other notions of equivalence of reductions, all going back to Lévy [1976]. An essential element is that one abstracts from the order in which (residuals of) redexes are contracted. For that reason one often speaks of *permutation equivalence*. Chapter 8 of Terese [2003] gives a thorough analysis of the various notions of equivalence of reductions. The definition we give here will there be named *projection equivalence*.

4.6.7. Definition. Two finite reductions ρ, σ in an orthogonal TRS are called *permutation equivalent* if (see Figure 4.20)

$$\rho/\sigma = \sigma/\rho = \varnothing$$

4.6.8. Examples. (i) In the CL-term $S(II)(S(II)(II))$ the three occurrences of the redex II can be contracted in any of six ways. All these six reductions are permutation equivalent.

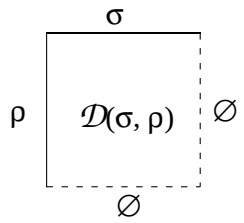


Figure 4.20: Permutation equivalent reductions

(ii) The two reductions of the CL-term $SKS(II)$ set out on the upper and left sides of the reduction diagram in Figure 4.21 are permutation equivalent. This boils down to the fact that the lower and right sides consist of empty steps.

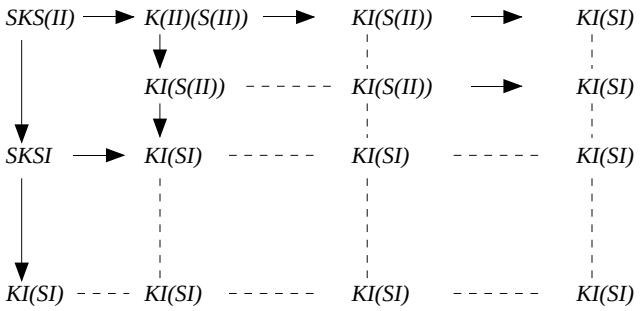


Figure 4.21: An example of permutation equivalence in CL

4.6.9. Exercise (Klop [1980]). Consider the completed reduction diagram as in Figure 4.22. Prove that two reduction sequences in this diagram with the same start point and the same endpoint (in the figure, the upper and lower boundaries of the shaded area) are permutation equivalent.

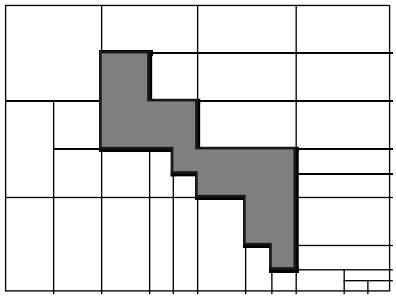


Figure 4.22: Equivalent reductions in diagram

4.7. Non-erasing reductions

Apart from confluence, many interesting facts can be proved for orthogonal TRSs. In this section we will show that under certain conditions the property of SN for an orthogonal TRS can be related to weaker notions. A key role is played by the Erasure Lemma, which uses the notion of *erasure*.

Recall from Definition 2.3.3 that a reduction rule $t \to s$ is *non-erasing* if the same variables occur in t and in s. A *TRS* is non-erasing if all its rules are. E.g. CL is not non-erasing (that is, it is *erasing*), due to the rule $Kxy \to x$. We also have the notion of an erasing *reduction step*, namely simply a step according to an erasing reduction rule.

4.7.1. Example. Just as the λ-calculus has a non-erasing version, called λI-calculus, there is a non-erasing version of combinatory logic. It is called CL_I, and has instead of I, K, S of CL the four basic combinators I, B, C, S with the following rules:

$$\begin{array}{lcl} Ix & \to & x \\ Bxyz & \to & x(yz) \\ Cxyz & \to & xzy \\ Sxyz & \to & xz(yz) \end{array}$$

CL_I is an orthogonal, non-erasing TRS. Another 'basis' for CL_I consists of the two constants I and J with rules

$$\begin{array}{lcl} Ix & \to & x \\ Jxyzw & \to & xy(xwz) \end{array}$$

(In Barendregt [1984] it is shown how to express $\{I, B, C, S\}$ in terms of $\{I, J\}$.) Remarkably, all J-terms, i.e. terms built by application from J alone, are strongly normalizing. See Probst and Studer [2001].

4.7.2. Definition. (i) We write $\infty(t)$ if t has an infinite reduction, that is, if $\neg\mathrm{SN}(t)$.

(ii) A reduction step $t \to t'$ is called *critical* if $\infty(t)$ but not $\infty(t')$.

So a critical step is one where the possibility of performing an infinite reduction is lost.

For the proof of the Erasure Lemma we need a simple proposition. The proof is left to the reader as an exercise.

4.7.3. Proposition (absorption). *Let t be a term in an orthogonal TRS, containing pairwise disjoint redexes $s_1, \ldots, s_n$, and a redex s. Let $t \twoheadrightarrow t'$ be the n-step reduction obtained by contraction, in some order, of the redexes $s_1, \ldots, s_n$. (So we have $t \twoheadrightarrow\!\!\!\!\!\!\cdot\; t'$.) Suppose s has after the reduction $t \twoheadrightarrow t'$ no residuals in t'.*

Then for some $i \in \{1, \ldots, n\}$ we have $s \subseteq s_i$. This means that s either coincides with some s_i ('coincidence') or is contained in an argument of some s_i ('erasure').

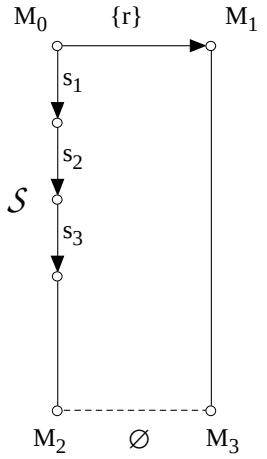


Figure 4.23: Absorption

4.7.4. Proposition (Erasure Lemma). *Let $M_0 \to_s M_0'$ be a critical step in an orthogonal TRS, where s is the contracted redex. Then the step $\to_s$ erases a subterm p of M_0 with $\infty(p)$.*

Proof. As $M_0 \to M_0'$ is a critical step, we have $\infty(M_0)$ and not $\infty(M_0')$. So M_0 has an infinite reduction $\mathcal{R}$, but M_0' has not. Hence, the projection $\mathcal{R}/\!\to_s$ must be the empty reduction $\varnothing$ from a certain stage onwards. Let $\mathcal{R} = M_0 \to_0 M_1 \to_1 M_2 \cdots$, with r_n the redex contracted in the step $M_n \to M_{n+1}$. Let $\mathcal{S}_n$ be the 'parallel move' contracting the residuals of s after $M_0 \twoheadrightarrow M_n$. So from a certain n onwards, r_n is absorbed by $\mathcal{S}_n$. By the previous proposition, this can happen by coincidence, or by erasure. Coincidence is possible only finitely many times, since after such an event, the resulting $\mathcal{S}_{n+1}$ contracts one redex less than $\mathcal{S}_n$. We conclude that from a certain N onwards, the redexes $r_N, r_{N+1}, \ldots$ are always absorbed by erasure (by one of the redexes in $\mathcal{S}_n$). See Figure 4.24.

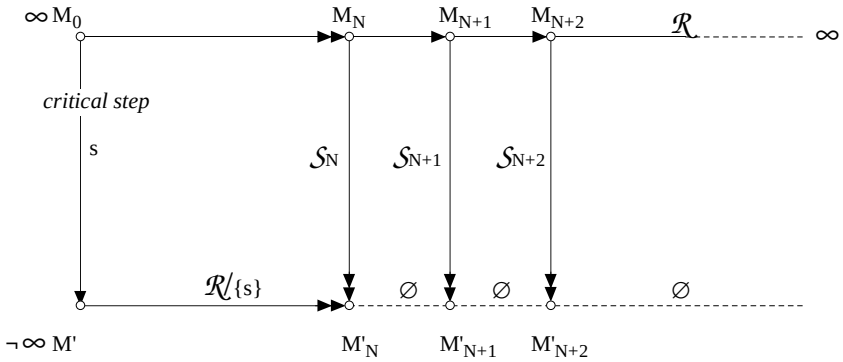


Figure 4.24: Erasure Lemma

Let the prefix C be the pattern of s. Then $s \equiv C[t_1, \ldots, t_n]$ and the

residuals of s in the parallel move $\mathcal{S}_n$ always have the form $C[t'_1, \ldots, t'_n]$, with the same pattern C and with $t_i \twoheadrightarrow t'_i$ $(i = 1, \ldots, n)$. Let us take a concrete example: $C \equiv F(G(\Box), \Box, H(\Box))$ is the pattern of the rule according to which s is a redex. So $s \equiv F(G(t_1), t_2, H(t_3))$. And $\mathcal{S}_N$ contracts the (say) k residuals of s: $F(G(t_{1,j}), t_{2,j}, H(t_{3,j}))$, for $j = 1, \ldots, k$. Likewise for all the subsequent $\mathcal{S}_{N+p}$ $(p \geq 0)$. We can visualize this as a sequence of blocks of triples $\Box\Box\Box$, as in Figure 4.25, each block having $3 \times k$ boxes $\Box$.

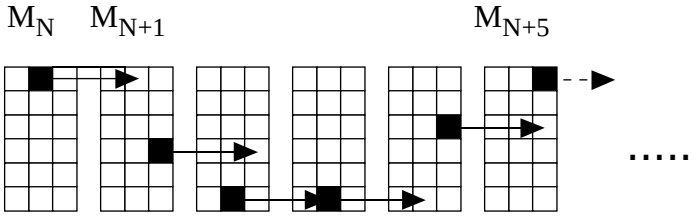


Figure 4.25: Infinite sequence of reduction steps

From each block to the next, there is a step in one of the boxes. Hence, by the Pigeon-Hole Principle one of the triples $\Box\Box\Box$ in the first block has to be infinitely many times 'active'; and of this triple again one box $\Box$. It follows that one of the original t_1, t_2, t_3 in s has the property ∞, say t_2. Now this t_2 has to be erased in the step $t_0 \to t'_0$, for otherwise also t'_0 would have an infinite reduction. □

The next theorem was proved in Church [1941] for the case of the non-erasing version of λ-calculus, the λI-calculus, where the restriction on term formation is adopted saying that in every abstraction term $\lambda x.M$ the variable x must have a free occurrence in M.

4.7.5. Theorem (Church [1941]). *Let $\mathcal{T}$ be orthogonal and non-erasing. Then $\mathcal{T}$ is WN if and only if $\mathcal{T}$ is SN.*

Proof. The implication SN $\Rightarrow$ WN is trivial. For the other direction, suppose there is a TRS $\mathcal{T}$ that is WN, but not SN. For a term t such that $\infty(t)$, consider a reduction to normal form. This reduction must contain a critical step, which, by the Erasure Lemma, has to be erasing. This contradicts the assumption that $\mathcal{T}$ is non-erasing. □

Another useful theorem, which also reduces the burden of a termination (SN) proof for orthogonal TRSs, uses the notion of innermost reduction.

4.7.6. Definition. (i) In an *innermost* reduction step a redex is contracted that contains no proper subredexes.

(ii) An *innermost* reduction is a sequence of innermost reduction steps.

(iii) A term t is *weakly innermost normalizing* (WIN) if it has a normal form which can be reached by innermost reduction from t.

(iv) A TRS is WIN if all its terms are.

4.7.7. THEOREM (O'Donnell [1977]). *Let $\mathcal{T}$ be an orthogonal TRS. Then $\mathcal{T}$ is WIN if and only if $\mathcal{T}$ is SN.*

PROOF. One direction is trivial again. The other direction follows from the Erasure Lemma by observing that an innermost contraction cannot erase a proper subterm which admits an infinite reduction, since otherwise the contracted redex would not have been innermost. □

The last two theorems can be refined to terms. The local version of Theorem 4.7.5 says that for a term in an orthogonal, non-erasing TRS the properties WN and SN coincide. Likewise there is a local version of Theorem 4.7.7. Thus, if in CL a term can be normalized via an innermost reduction, all its reductions are finite.

Another illustration of the use of the Erasure Lemma is the simple proof of the modularity of SN for orthogonal TRSs (Theorem 5.7.5).

4.7.8. EXERCISE (Khasidashvili [1990]). Define the TRS $\mathcal{T}$ to be *completely normalizing* (CN) if each $t \in Ter(\mathcal{T})$ can be normalized by means of redex contractions in which no redexes are erased (so only subterms in normal form may be erased).

Prove the equivalence: $\mathcal{T}$ is CN if and only if $\mathcal{T}$ is SN.

4.7.9. REMARK. Semi-Thue systems, viewed as TRSs as explained in Subsection 3.3.4, are non-erasing (since both left-hand side and right-hand side of the TRS version of a rule always end in x). Also, if there are no critical pairs in the semi-Thue system, it is orthogonal. So if a semi-Thue system has no critical pairs, the properties SN and WN coincide.

This rather trivial observation could have been more easily made by noting that for a semi-Thue system without critical pairs the property CR^1 holds, as defined in Exercise 1.3.8, whence WN if and only if SN.

4.7.10. EXERCISE (Klop [1980]). Prove that for all orthogonal TRSs $\mathcal{T}$ with finite alphabet and finitely many reduction rules we have the following (FB stands for *finitely branching* and Inc for *increasing*, see Definition 1.1.15):

(i) $\mathcal{T}$ is non-erasing if and only if $\leftarrow$ has the property FB;

(ii) $\leftarrow$ has the property SN if and only if $\rightarrow$ has the property Inc.

4.7.11. EXERCISE. Let $\mathcal{T}$ be an orthogonal TRS with a signature only containing constants. So $\mathcal{T}$ is in fact an ARS. Note that $\mathcal{T}$ is, trivially, non-erasing. Explain why for such TRSs the equivalence between SN and WN is trivial.

4.7.12. EXERCISE. Show that (the local version of) O'Donnell's Theorem does not hold for λ-calculus. That is, give a λ-term that is WIN but not SN.

4.8. Reduction strategies

Terms in a TRS may have a normal form as well as admitting infinite reductions. So, if we are interested in finding normal forms, we should have some strategy at our disposal telling us what redex to contract in order to achieve that desired result. We will in this section present some strategies which are guaranteed to find the normal form of a term whenever such a normal form exists.

We will adopt the restriction to orthogonal TRSs (except for the definition of reduction strategy).

The strategies below will be of two kinds: *one-step* strategies, which point in each reduction step to just one redex as the one to contract, and *many-step* strategies. In the many-step strategies we consider in this chapter, always a set of redexes is contracted simultaneously. Of course all strategies must be computable.

Apart from the objective of finding a normal form, we will consider the objective of finding a 'best possible' reduction even if the term at hand does not have a normal form.

4.8.1. Definition. Consider a TRS $\mathcal{T}$.

(i) A *one-step reduction strategy* $\mathbb{F}$ *for* $\mathcal{T}$ is a mapping $\mathbb{F} : Ter(\mathcal{T}) \to Ter(\mathcal{T})$ such that

- $t \equiv \mathbb{F}(t)$ if t is a normal form,
- $t \to \mathbb{F}(t)$ otherwise.

(ii) A *many-step reduction strategy* $\mathbb{F}$ *for* $\mathcal{T}$ is a mapping $\mathbb{F} : Ter(\mathcal{T}) \to Ter(\mathcal{T})$ such that

- $t \equiv \mathbb{F}(t)$ if t is a normal form,
- $t \to^+ \mathbb{F}(t)$ otherwise.

Here $\to^+$ is the transitive (but not reflexive) closure of $\to$.

4.8.2. Definition. (i) A reduction strategy (one-step or many-step) $\mathbb{F}$ for $\mathcal{T}$ is *normalizing* if for each term t in $Ter(\mathcal{T})$ having a normal form, the sequence $\{\mathbb{F}^n(t) \mid n \geq 0\}$ contains a normal form.

(ii) $\mathbb{F}$ is *cofinal* if for each t the sequence $\{\mathbb{F}^n(t) \mid n \geq 0\}$ is cofinal in $\mathcal{G}(t)$, the reduction graph of t. (See Definition 1.1.15 for 'cofinal' and see Figure 4.26.)

Trivially we have

4.8.3. Proposition. *Cofinal strategies are normalizing.*

4.8.4. Remark. A normalizing reduction strategy is good, but a cofinal one is even better: it finds, when applied to term t, the best possible reduction sequence

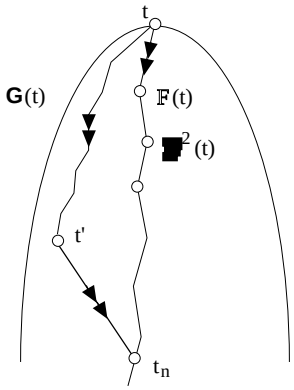


Figure 4.26: A cofinal reduction

starting from t (or rather, a best possible) in the following sense. Consider a reduction $t \to s$ as a gain in information; thus normal forms have maximum information. In case there is no normal form in $\mathcal{G}(t)$, one can still consider infinite reductions as developing more and more information. (See Figure 4.26.) Now the cofinal reductions $t \equiv t_0 \to t_1 \to t_2 \to \cdots$ are optimal since for every t' in $\mathcal{G}(t)$ they contain a t_n with information content no less than that of t' (since $t' \twoheadrightarrow t_n$ for some t_n, by definition of cofinal).

We now present the most usual reduction strategies. In all the five strategies given, the criterion for redex selection in some way makes use of the positions of the redexes in the term. Therefore these strategies might be called *positional*.

4.8.5. Definition. (i) The *leftmost-innermost* (one-step) strategy is the strategy in which in each step the leftmost of the minimal or innermost redexes is contracted.

(ii) The *parallel-innermost* (many-step) strategy reduces simultaneously all innermost redexes. Since these are pairwise disjoint, this is no problem.

(iii) The *leftmost-outermost* (one-step) strategy: in each step the leftmost redex of the maximal (or outermost) redexes is reduced. Notation: $\mathbb{F}_{\mathrm{lm}}$.

(iv) The *parallel-outermost* (many-step) strategy reduces simultaneously all maximal redexes; since these are pairwise disjoint, this is no problem. Notation: $\mathbb{F}_{\mathrm{po}}$.

(v) The *full-substitution rule* (or Kleene reduction, or Gross–Knuth reduction): this is a many-step strategy in which all redexes are simultaneously reduced. Notation: $\mathbb{F}_{\mathrm{GK}}$. More precisely: $\mathbb{F}_{\mathrm{GK}}$ is the result of the complete development of the set of all redexes in t.

These five strategies are illustrated in Figure 4.27, using the reduction rules of Table 4.1.

$\wedge(\top, x)$	$\to$	x
$\wedge(\bot, x)$	$\to$	$\bot$
$\vee(\top, x)$	$\to$	$\top$
$\vee(\bot, x)$	$\to$	x

Table 4.1: The rules used in Figure 4.27

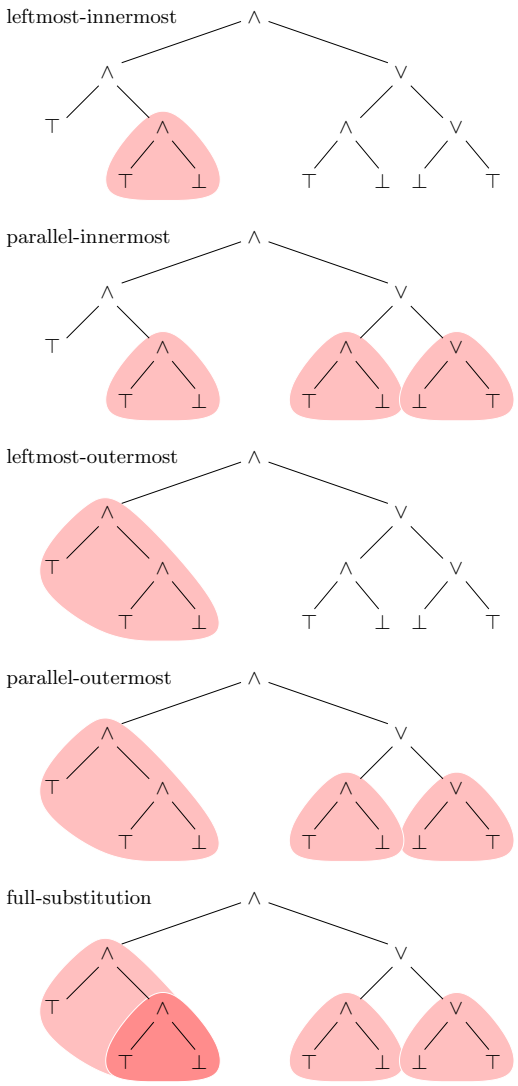


Figure 4.27: Five reduction strategies (see Table 4.1)

Note that the positional strategies of Definition 4.8.5 are only defined for orthogonal TRSs. For example, one cannot select a rule to apply to a in the ambiguous TRS $\{a \to b, a \to c\}$ by just indicating a redex position.

We will be mainly interested here in the strategies (iii)–(v), for a reason that will be clear by inspection of Table 4.2 below.

4.8.6. Remark. For λ-calculus $\mathbb{F}_{\mathrm{lm}}$ is also a normalizing strategy, just as it is for the orthogonal TRS CL (combinatory logic). However, in general $\mathbb{F}_{\mathrm{lm}}$ is not a normalizing strategy for orthogonal TRSs:

4.8.7. Examples. (i) An example showing that the leftmost-outermost strategy is not normalizing in general is given in Huet and Lévy [1991]. Take the orthogonal TRS $\{F(x,B) \to D, A \to B, C \to C\}$ and consider the term $F(C,A)$. This term has a normal form which is not found by the leftmost-outermost strategy.

(ii) An example showing that parallel-outermost reduction need not be cofinal can be found in CL (see Table 3.3). That is, define the term I_t as SKt, and check that $I_t x \twoheadrightarrow x$. Furthermore, define the term Ω_t as $SI_tI_t(SI_tI_t)$. Now the parallel-outermost strategy, applied to Ω_{II}, yields a cyclic reduction sequence $\Omega_{II} \twoheadrightarrow \Omega_{II}$ which is not cofinal since $\Omega_{II} \twoheadrightarrow \Omega_I$ but not $\Omega_I \twoheadrightarrow \Omega_{II}$.

Even though $\mathbb{F}_{\mathrm{lm}}$ is in general for orthogonal TRSs not normalizing, there is a large class of orthogonal TRSs for which it is: the ones that are *left-normal*.

4.8.8. Definition (O'Donnell [1977]). An orthogonal TRS is *left-normal* if in every reduction rule $t \to s$ the constant and function symbols in the left-hand side t precede (in the linear term notation) the variables.

4.8.9. Examples. (i) Combinatory logic is left-normal. This holds for all the usual 'bases' as in Example 4.7.1.

(ii) Recursive program schemes as defined in Definition 3.3.7 are left-normal.

(iii) $F(x,B) \to D$ is not left-normal; $F(B,x) \to D$ is left-normal.

Using the Standardization Theorem for left-normal orthogonal TRSs, see Chapter 4 of Terese [2003], one can prove that $\mathbb{F}_{\mathrm{lm}}$ is normalizing for such TRSs.

4.8.1. Relaxing the constraints in $\mathbb{F}_{\mathrm{lm}}$, $\mathbb{F}_{\mathrm{GK}}$ and $\mathbb{F}_{\mathrm{po}}$

In the reduction strategy $\mathbb{F}_{\mathrm{GK}}$ (full substitution) every redex is 'killed' as soon as it arises, and this repeatedly. Suppose we relax this requirement, and allow ourselves some time (i.e. some number of reduction steps) before getting rid of a particular redex – but with the obligation to deal with it eventually. The reductions arising in this way are all cofinal.

4.8.10. Definition. Let $\rho = t_0 \to t_1 \to \cdots$ be a finite or infinite reduction sequence.

(i) Consider some redex s in some term t_n of ρ. We say that s is *secured* in ρ if eventually there are no residuals of s left, i.e. for some $m > n$ the term t_m (and hence each t_k with $k \geq m$) contains no residual of s.

(ii) The reduction ρ is *fair* if every redex occurring in a term in ρ is secured.

4.8.11. REMARK. Note that we have the following simple fact. Let $\sigma : t \twoheadrightarrow t'$ be a finite reduction, and $\rho : t' \to t'' \to \cdots$ be an infinite reduction starting where σ left off. Then the concatenation $\sigma \cdot \rho$ is fair iff ρ is fair.

4.8.12. THEOREM. *For reductions ρ in orthogonal TRSs we have: ρ is fair $\Rightarrow$ ρ is cofinal.*

PROOF. For finite fair reductions the theorem is trivial; these are normalizing. So consider the infinite fair reduction

$$\rho : t_0 \equiv s_0 \twoheadrightarrow s_1 \twoheadrightarrow s_2 \twoheadrightarrow \cdots$$

and the reduction $t_0 \to t_1 \to \cdots \to t_n$. (See Figure 4.28.) Let r_i be the redex contracted in the step $t_i \to t_{i+1}$ for $i = 0, \ldots, n-1$. Because ρ is fair and by the Parallel Moves Lemma, the step $t_0 \to t_1$ is absorbed by ρ (that is, 'pushing $t_0 \to t_1$ through' ρ yields eventually the empty reduction $\varnothing$). Let this be the case at the term s_{i_0} in ρ. So $\{r_0\}/\{s_0 \twoheadrightarrow s_{i_0}\} = \varnothing$. Now the projection $\rho' = \rho/\{t_0 \to t_1\}$ is again a fair reduction, by Remark 4.8.11, since ρ and ρ' coincide from s_{i_0} onwards. So we apply the same procedure to the second step $t_1 \to t_2$, which is absorbed by ρ at the term s_{i_1}, that is, $\{r_1\}/\{t_1 \twoheadrightarrow s_{i_1}\} = \varnothing$. Continuing in this way we find a term s_{i_k} in ρ such that the whole vertical reduction $t_0 \twoheadrightarrow t_n$ is absorbed at that point, i.e. $\{t_0 \twoheadrightarrow t_1\}/\{s_0 \twoheadrightarrow s_{i_k}\} = \varnothing$. This means that $t_n \twoheadrightarrow s_{i_k}$, which proves that the sequence ρ is indeed cofinal. □

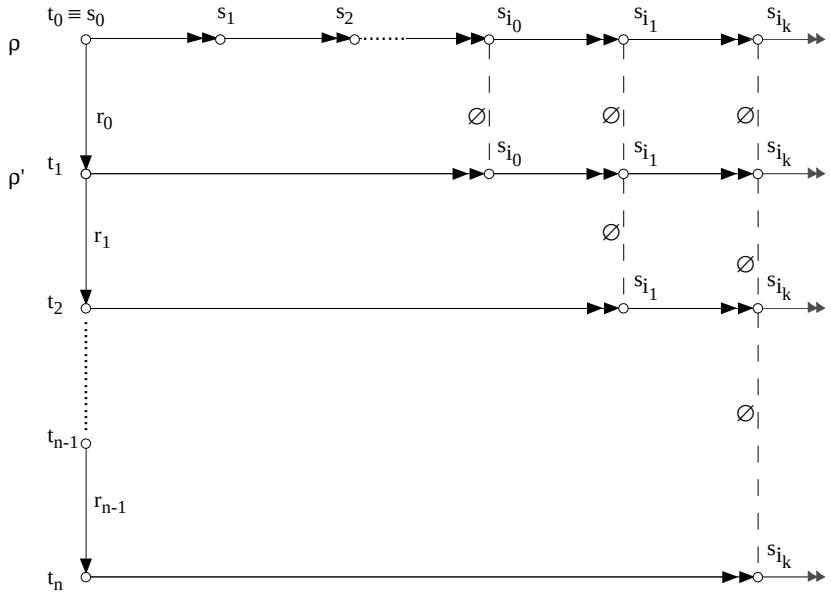


Figure 4.28: Fair implies cofinal

4.8.13. Remark. One easily sees that a reduction ρ is fair if and only if there is no redex in a term of ρ 'staying alive for ever', in the sense of having an infinite chain of residuals. Then, by König's Lemma, a reduction ρ is fair if and only if every tree of residuals in ρ is finite.

4.8.14. Corollary. *For orthogonal TRSs, $\mathbb{F}_{\mathrm{GK}}$ is a cofinal (and hence also normalizing) reduction strategy.*

Proof. In each $\mathbb{F}_{\mathrm{GK}}$-step all redexes are contracted; so the reduction delivered by $\mathbb{F}_{\mathrm{GK}}$ is fair. □

A similar relaxation of constraints applies to the other two strategies $\mathbb{F}_{\mathrm{po}}$ and $\mathbb{F}_{\mathrm{lm}}$.

4.8.15. Definition. (i) A reduction ρ is *leftmost-fair* if either ρ ends in a normal form, or infinitely many times the leftmost-outermost redex is contracted in ρ.

(ii) A reduction $\rho = t_0 \to t_1 \to \cdots$ is *outermost-fair* if ρ does not contain a term t_n with an outermost redex which infinitely long stays an outermost redex but which is never contracted.

A more exact definition of the notion of outermost-fairness can be given analogously to Definition 4.8.10.

	orthogonal	orthogonal left-normal	orthogonal non-erasing
leftmost–innermost	p	p	p n
parallel–innermost	p	p	p n
leftmost–outermost (leftmost–fair)		n	p n
parallel–outermost (outermost–fair)	n	n	p n
full substitution (fair)	n c	n c	p n c

Table 4.2: Some properties of strategies

The main properties of the various reduction strategies (and their relaxed versions) are summarized in Table 4.2. The letters p, n, c stand for perpetual, normalizing, cofinal respectively. In case a property is not mentioned, it does not hold generally. Note that for the leftmost-outermost strategy, when applied to orthogonal TRSs in general, none of the three properties holds generally.

The new notion of a perpetual strategy, which is mentioned in the table, is defined as follows.

4.8.16. Definition. A reduction strategy $\mathbb{F}$ for $\mathcal{T}$ is *perpetual*, if for all t we have $\infty(t) \Rightarrow \infty(\mathbb{F}(t))$.

So a perpetual strategy is the opposite of a normalizing one; it tries to avoid normal forms whenever possible, and could therefore also be called 'anti-normalizing'.

5

Properties of rewriting: decidability and modularity

Jan Willem Klop
Roel de Vrijer

This chapter has two main topics, each one presenting a part of the basic theory of first-order term tewriting systems. In the first sections the issue of decidability of properties of rewriting, such as confluence and termination is addressed. Special attention is given to CL. Then the focus shifts to the question whether, and under what conditions, properties such as confluence and termination are preserved when term rewriting systems with these properties are combined. This subject is called modularity.

5.1. Decidability

This section is devoted to decidable and undecidable properties of term rewriting systems. Typical properties for which we would like to have decision algorithms are for example strong normalization and confluence. But one can also ask for example whether two terms are convertible, whether a term has a normal form, or whether a term has an infinite reduction.

We restrict ourselves to TRSs with a finite signature and a finite (or decidable) set of reduction rules.

Decidability problems can be posed in several ways. First there is the level of terms:

> Given a property P, like SN or CR, is there an algorithm that takes an arbitrary TRS $\mathcal{R}$ and a term t of $\mathcal{R}$ as input and decides whether t satisfies property P?

If this is the case we call P a *decidable* property of TRSs. Decidabilitity in this sense might also be called *local* decidability, for better contrast with the stronger notion of uniform decidability to be defined next.

We say that a *TRS* has property P (e.g. SN) if all its terms have property P. Now the decision problem can be lifted to the level of TRSs:

> Is there an algorithm that takes a TRS $\mathcal{R}$ as input and decides whether $\mathcal{R}$ satisfies property P?

If this is the case we call P *uniformly decidable*.

A similar distinction exists between the *halting problem* and the *uniform halting problem* for Turing machines. In the halting problem one asks for an algorithm that, given a Turing machine M with input tape T, decides whether M will halt with input T. In contrast, the uniform halting problem asks for an algorithm that decides whether an arbitrary Turing machine will halt on *every* input tape. As a matter of fact, there will turn out to be also a technical connexion: in Subsection 5.3.1 the undecidability of the halting problem and the uniform halting problem for Turing machines will be used in proofs of, respectively, the undecidability and the uniform undecidability of SN.

The standard procedure for showing a property P of TRSs to be undecidable proceeds as follows. Choose a decision problem U that is known to be undecidable, such as for example the halting problem for Turing machines or Post's correspondence problem. Then show that if there were a decision procedure for the problem P at stake, that procedure could be used in devising a decision procedure for U. This is called *reducing* the problem U to P. More elaborate treatments of this matter, and examples of well-known undecidable problems, can be found in textbooks on recursion theory and on the theory of computability and automata, for example Rogers [1967] or Hopcroft et al. [2001].

Decidability questions can also be asked for a restricted class of TRSs, for example ground TRSs. In general, restriction of the class of TRSs gives rise to so-called *relative* (un)decidability results, as in Geser et al. [1997].The extreme case is to consider just *one* fixed TRS, say $\mathcal{R}$. Then only the local decidability problem remains: is there an algorithm deciding P for arbitrary input of a term t in $\mathcal{R}$? In Section 5.4 we will study the case of CL.

Overview

The properties strong normalization and Church–Rosser are undecidable, both uniformly and on the term level. For CR this will be proved in Section 5.2. For SN several alternative variants of an undecidability proof are given in Section 5.3.

5.2. Easy (un)decidability results

In this section we cover some results that are easy to obtain. We start with the undecidability of confluence.

5.2.1. Theorem. CR *and* WCR *are undecidable properties of TRSs.*

Proof. Table 5.1 displays the set of equations E of an equational specification $\mathcal{E}$ of a simple semi-group with undecidable word problem from Matijasevich [1967].

$x(yz)$	$=$	$(xy)z$
$abaabb$	$=$	$bbaaba$
$aababba$	$=$	$bbaaaba$
$abaaabb$	$=$	$abbabaa$
$bbbaabbaaba$	$=$	$bbbaabbaaaa$
$aaaabbaaba$	$=$	$bbaaaa$

Table 5.1: Semi-group with undecidable word problem

Take all rewrite rules $\rightarrow$ that can be made from these equations, in both directions. This yields the set of rewrite rules $E^{\leftrightarrow}$. Define the TRS $R(s,t) = E^{\leftrightarrow} \cup \{A \rightarrow s, A \rightarrow t\}$ where A is some new constant. Now $R(s,t)$ is confluent if and only if $s =_E t$, if and only if $R(s,t)$ is WCR. □

As a matter of fact, the above proof also gives us the undecidability of the property ground-CR. Which in this case boils down to CR for $(\mathcal{E}^{\leftrightarrow})_0$, the TRS $\mathcal{E}^{\leftrightarrow}$ restricted to ground terms. By contrast, for *ground TRSs*, i.e., TRSs where in every rule $t \rightarrow s$ the terms t, s are ground terms (not to be confused with $(\mathcal{R})_0$ above), both confluence and termination are decidable. A decidability proof of CR for ground TRSs does not fit in this section with easy results. In fact it is quite complicated. We refer to Dauchet and Tison [1984], Dauchet et al. [1990], Oyamaguchi [1987].

With the property SN we have the same situation. Ground-SN is undecidable, but SN for ground TRSs is decidable. A proof of the latter statement is covered in the next exercise.

5.2.2. Exercise (Huet and Lankford [1978]). Let $\mathcal{R}$ be a finite ground TRS. A term t is called *self-embedding*, if $t \rightarrow^+ C[t]$ for some context $C[\,]$.

(i) Prove that for an arbitrary TRS $\mathcal{R}$ we have that $\mathcal{R}$ is SN iff all terms that are redexes according to $\mathcal{R}$ are SN, iff all contracta of such redexes are SN.

(ii) Let $\mathcal{R}$ be a ground TRS. Then, if $\mathcal{R}$ is not SN, some contractum of an $\mathcal{R}$-redex admits an infinite reduction.

(iii) Now let $\mathcal{R}$ be a finite ground TRS. Then, if $\mathcal{R}$ is not SN, some redex is self-embedding.

(iv) Again let $\mathcal{R}$ be a finite ground TRS. Then it is decidable whether $\mathcal{R}$ is SN.

5.3. Undecidability of strong normalization

The undecidability of SN is proved by reducing a problem that is already known to be undecidable to the problem of deciding SN. There are several options for choosing such a standard undecidable problem. We will sketch some in this section. The halting problem for Turing machines is used in Subsection 5.3.1. Then in Subsection 5.3.2 we translate the undecidable Post correspondence problem into a question of termination of TRSs.

One reason for giving these alternative treatments of the same result is that they present interesting case studies in using the term rewriting format to model various notions of computation.

5.3.1. Undecidability of SN via Turing machines

In this section we prove that SN is an undecidable property for TRSs, via a translation of the problem to the (uniform) halting problem for Turing machines. The presentation is based on an exercise in Klop [1992]. The basic idea is the same as in the original Huet and Lankford [1978]. However, we use a different encoding of Turing-machine configurations as TRS-terms.[1] We will not be concerned with the exact number of reduction rules employed in the translation of a Turing machine to a TRS. An undecidability proof using a TRS of only two reduction rules, thus establishing that SN is undecidable even for TRSs with only two rules, is given by Dershowitz [1987]. This result was subsequently strengthened to one-rule TRSs by Dauchet [1989] – the one-rule TRS can even be an orthogonal one.

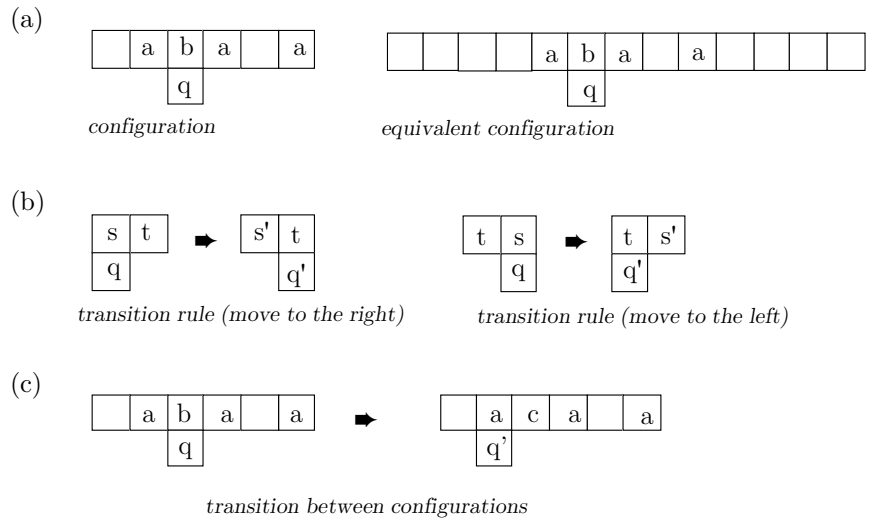


Figure 5.1: Turing machine configurations and transitions

5.3.1. Definition. A *deterministic Turing machine* M consists of a triple $\langle Q, S, \delta \rangle$, where

(i) Q is a finite set $\{q_0, \ldots, q_n\}$ of *states*;

(ii) $S = \{\Box, s_1, \ldots, s_m\}$ is the set of *tape symbols*, $\Box$ being the *empty symbol* or *blank*, $Q \cap S = \varnothing$;

[1] A side effect of encoding used here is that, whereas Huet and Lankford's proof works also for string rewriting systems, the proof in this section does not.

(iii) δ is a partial function from $Q \times S$ to $Q \times S \times \{L, R\}$, called the *transition function*. Here L represents a move to the left, R a move to the right.

A Turing machine is supposed to operate on a tape which is infinite in two directions, with only finitely many positions filled with non-empty symbols. More precisely, the tape can be thought of as a function $\tau : \mathbb{Z} \to S$, such that $\{i \in \mathbb{Z} \mid \tau(i) \neq \Box\}$ is finite. At each particular instant the machine is in a certain state, say q, and reads a position p on the tape, say containing symbol s; this is called a *configuration*. The transition function then determines what happens next, i.e., what will be the next configuration: if $\delta(q, s) = \langle q', s', R\rangle$, the machine state changes to q', the symbol s is replaced by s' and the machine reads the position immediately right of p (left if we had L instead of R). This goes on either ad infinitum, or until a symbol s is read while the machine is in a state q, such that $\delta(q, s)$ is not defined – the machine is then said to *halt*.

Configurations can be depicted as in Figure 5.1(a), the square containing the state q is attached to the position of the tape that is read. The (infinite) part of the tape that is not shown is supposed to contain only blank symbols. Therefore the two configurations shown in Figure 5.1(a) are in fact the same. Figure 5.1(b) depicts the transition rules (a right move and a left move) that, respectively, correspond to values $\delta(q, s) = \langle q', s', R\rangle$ and $\delta(q, s) = \langle q', s', L\rangle$ of the transition function.

Often strings are used to encode Turing-machine configurations. A string of the form $w_1 q w_2$, with $w_1, w_2 \in S^*, q \in Q$, called an *instantaneous description*, codes the machine in state q, with tape $w_1 w_2$ (the rest blank), the head scanning the first symbol to the right of q, that is, the first symbol of w_2, or $\Box$ if w_2 is empty. For example, the configurations in Figure 5.1(a) correspond to the (equivalent) instantaneous descriptions $\Box aqba\Box a$ and $\Box\Box\Box\Box aqba\Box a\Box\Box\Box\Box$.

We can represent a Turing machine as a term rewriting system. One way would be to make use directly of the instantaneous descriptions in the form of strings; which would naturally give rise to a string rewriting system (see Subsection 3.3.4). Instead, we use a different representation, coding a state q as a binary function symbol.

To the Turing machine $M = \langle Q, S, \delta\rangle$ we associate a TRS $\mathcal{R}_M$ as follows. For each $q \in Q$ there is a binary function symbol which we will denote by the same letter. Each $s \in S$ corresponds to a unary function symbol, also denoted by the same letter. Furthermore, the alphabet of $\mathcal{R}_M$ contains a constant symbol $\triangleright$, which should be thought of as an infinite stretch of blanks. A word $w \in S^*$ is translated into the term $\phi(w)$ as follows:

$$\begin{array}{lcll} \phi(\varepsilon) & = & \triangleright, & \text{where } \varepsilon \text{ is the empty word} \\ \phi(sw) & = & s(\phi(w)), & \text{for } s \in S, w \in S^* \end{array}$$

For example, the translation of $ba\Box a$ is $b(a(\Box(a(\triangleright))))$. It is convenient to suppress the parentheses that we just introduced again, by association to the

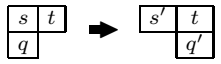
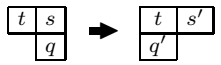
(L/R)-move	transition	rewrite rules
$\delta(q,s) = \langle q', s', R\rangle$		$q(x,sy) \rightarrow q'(s'x,y)$
$\delta(q,s) = \langle q', s', L\rangle$		$q(tx,sy) \rightarrow q'(x,ts'y)$

Table 5.2: Main reduction rules of the TRS $\mathcal{R}_M$

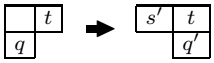
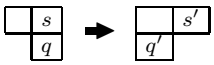
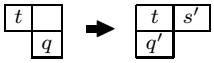
(L/R)-move	transition	extra rewrite rules
$\delta(q,\Box) = \langle q', s', R\rangle$		$q(x,\rhd) \rightarrow q'(s'x,\rhd)$
$\delta(q,s) = \langle q', s', L\rangle$		$q(\rhd,sy) \rightarrow q'(\rhd,\Box s'y)$
$\delta(q,\Box) = \langle q', s', L\rangle$		$q(tx,\rhd) \rightarrow q'(x,ts'\rhd)$ $q(\rhd,\rhd) \rightarrow q'(\rhd,\Box s'\rhd)$

Table 5.3: Extra rules of $\mathcal{R}_M$ because of $\rhd = \Box\rhd$

right. Thus the term $b(a(\Box(a(\rhd))))$, representing the string $ba\Box a$, is now rendered as $ba\Box a\rhd$.

An instantaneous description w_1qw_2 will now be translated to the $\mathcal{R}_M$-term $q(\phi(w_1^{-1}),\phi(w_2))$, where w_1^{-1} is w_1 reversed. For example, the configuration $\Box aqba\Box a$ is translated to $q(a\Box\rhd, ba\Box a\rhd)$. Note that in this representation the machine scans the head symbol of the right argument of q. The equivalence we mentioned between instantaneous descriptions, adding or deleting blanks on the left or the right, makes configurations of M again not uniquely represented by $\mathcal{R}_M$-terms. One has to work modulo the identity $\rhd = \Box\rhd$.

We now translate the transition rules of M into reduction rules of $\mathcal{R}_M$. By our explanation of the encoding it is clear that to a transition rule of the R-type corresponds a reduction rule $q(x,sy) \rightarrow q'(s'x,y)$, and to a rule of the L-type corresponds a rule $q(tx,sy) \rightarrow q'(x,ts'y)$. These rules are given in Table 5.2.

However, since in the encoding the symbol $\rhd$ contains hidden $\Box$s, we need to add four extra rules. These extra rules are given in Table 5.3. Intuitively, they can be derived by using the "tape equation" $\rhd = \Box\rhd$ and applying the main reductions in Table 5.2. For example, in the second rule $q(tx,sy) \rightarrow q'(x,ts'y)$ we can match tx with $\Box\rhd$, obtaining $q(\Box\rhd,sy) \rightarrow q'(\rhd,\Box s'y)$, while on the other hand $q(\Box\rhd,sy) = q(\rhd,sy)$. Thus we added the extra rule $q(\rhd,sy) \rightarrow q'(\rhd,\Box s'y)$. Adding the extra rules amounts to taking some kind of completion of the main rules with respect to the equation $\rhd = \Box\rhd$.

Now in the following exercise the TRS representation of Turing machines is used to reduce the halting problem for Turing machines to the decision problem for SN.

5.3.2. EXERCISE. (i) Prove that for Turing-machine configurations α, β we have

(1) If $\alpha \blacktriangleright \beta$ and t represents α, then for some term s we have $t \to s$ and s represents β.

(2) If t represents α and $t \to s$, then there is a β such that $\alpha \blacktriangleright \beta$ and s represents β.

(One might express this by saying that the relation "represents" between configurations and terms is a *bisimulation* with respect to transition on the Turing-machine side and reduction on the TRS-side.

(ii) Prove that, given a TRS $\mathcal{R}$ and a term t in $\mathcal{R}$, the problem of determining whether t has an infinite reduction in $\mathcal{R}$ is undecidable. This means there is no algorithm that accepts as inputs pairs $(\mathcal{R}, t)$ of a TRS $\mathcal{R}$ (given by a finite set of rewrite rules) and a term $t \in \mathit{Ter}(\mathcal{R})$, and that yields as output the answer 'yes' if t has an infinite reduction in $\mathcal{R}$, and 'no' otherwise. (Using (i), reduce this problem to the well-known undecidable halting problem for Turing machines with empty tape as initial configuration.)

(iii) To each ground term in $\mathcal{R}_M$ of the form $q(t_1, t_2)$ where t_1, t_2 are terms in which no $q' \in Q$ occurs (call such a term 'restricted'), there corresponds a configuration of M. This restriction is necessary: an example of a term t not corresponding to a configuration of M is $q(a\Box b\rhd, aq(a\rhd, b\rhd)b\rhd)$. Prove that if some term t in $\mathcal{R}_M$ has an infinite reduction in $\mathcal{R}_M$, then there is also a restricted ground term t' in $\mathcal{R}_M$ having an infinite reduction, and thus yielding a corresponding infinite run of the Turing machine M.

(iv) Prove, using (iii) and referring to the well-known undecidable *uniform* halting problem for Turing machines, that the problem of determining whether a given TRS is SN (strongly normalizing) is undecidable. The uniform halting problem for Turing machines is the problem of deciding whether a given Turing machine halts on every input as initial configuration.

5.3.2. Reducing Post's correspondence problem to SN

Post's correspondence problem (PCP) is the following problem. Given a finite alphabet Σ and a finite sequence $I = (\langle \alpha_i, \beta_i \rangle \,|\, i = 1, \ldots, n)$ of ordered pairs of non-empty words over Σ, does I have a *solution*? Here a solution is a sequence $i_1, \ldots, i_m$ $(1 \leq i_k \leq n)$ of indices, such that $\alpha_{i_1} \ldots \alpha_{i_m} \equiv \beta_{i_1} \ldots \beta_{i_m}$. I.e., such that the concatenation of the words $\alpha_{i_1}, \ldots, \alpha_{i_m}$ and the concatenation of the words $\beta_{i_1}, \ldots, \beta_{i_m}$ are the same word.

5.3.3. EXAMPLE. Let $\Sigma = \{0, 1\}$.

(i) $I = (\langle 01, 0\rangle, \langle 0, 10\rangle)$. The sequence $1, 2$ is a solution with match $01\ 0 \equiv 0\ 10$.

(ii) $I = (\langle 111, 0\rangle, \langle 00, 1\rangle)$. Now there is no solution.

It is a well-known classic result from recursion theory that it is undecidable whether an instance of PCP has a solution. See Post [1946]. As a matter of fact, PCP is already an unsolvable problem when we restrict ourselves to instances over the alphabet $\Sigma = \{0, 1\}$.

The undecidability of PCP can be used to prove the (uniform) undecidability of SN. The proof proceeds by a reduction of PCP to the SN problem for TRSs. To each instance I of PCP over $\Sigma = \{0, 1\}$ a TRS $\mathcal{R}_I$ is associated, in such a way that I has a solution if and only if $\mathcal{R}_I$ is not SN.

5.3.4. Definition. Let $I = \{\langle \alpha_i, \beta_i \rangle \mid i = 1, \ldots, n\}$ be an instance of PCP. Define a TRS $\mathcal{R}_I$ as follows. The signature consists of a constant $\Box$, unary function symbols $0, 1$, and a ternary function symbol P. If t is a term and α is a word over $\{0, 1\}$, then by $\alpha(t)$ we denote a term as we did in Example 3.3.4. The rewrite rules are the following.

$$\begin{array}{rlcll} \rho_i: & P(\alpha_i(x), \beta_i(y), z) & \to & P(x, y, z) & \text{for all pairs } \langle \alpha_i, \beta_i \rangle \in I \\ \rho_{\Box,0}: & P(\Box, \Box, 0(x)) & \to & P(0(x), 0(x), 0(x)) & \\ \rho_{\Box,1}: & P(\Box, \Box, 1(x)) & \to & P(1(x), 1(x), 1(x)) & \end{array}$$

As to the rules $\rho_{\Box,0}, \rho_{\Box,1}$, we want that $P(\Box, \Box, \gamma(\Box)) \to P(\gamma(\Box), \gamma(\Box), \gamma(\Box))$ for each non-empty word γ. But we do not want the side-effect $P(\Box, \Box, \Box) \to P(\Box, \Box, \Box)$, which would result from the simpler rule $\rho_\Box : P(\Box, \Box, x) \to P(x, x, x)$. With that rule the TRS $\mathcal{R}_I$ would be non-terminating without more.

In the proof of the next theorem we will make use of the following observation. Recall that the notation $t \rhd t'$ was introduced in Exercise 2.3.11 to denote that either $t \to t'$ or $t' \subset t$, with $\trianglerighteq$ as the reflexive closure of $\rhd$.

5.3.5. Remark. Let $P(s_1, s_2, s_3) \to P(s'_1, s'_2, s'_3)$ be a reduction step in $\mathcal{R}_I$, not the result of an application of one of the rules $\rho_{\Box,0}$ and $\rho_{\Box,1}$ at the root. Then either it is a ρ_i-step at the root, in which case s'_1, s'_2 are proper subterms of s_1, s_2 respectively; or the contracted redex is within one of the s_i. In either case we have $s_i \trianglerighteq s'_i$, $i = 1, 2, 3$, and at least one of these relations is proper: $s_i \rhd s'_i$.

5.3.6. Proposition. *An instance $I = \{\langle \alpha_i, \beta_i \rangle \mid i = 1, \ldots, n\}$ of PCP has a solution if and only if $\mathcal{R}_I$ is not SN.*

Proof. ($\Rightarrow$) Suppose I has a solution $\alpha_{i_1} \ldots \alpha_{i_m} \equiv \beta_{i_1} \ldots \beta_{i_m}$, and let $\gamma \equiv \alpha_{i_1} \ldots \alpha_{i_m}$. Then in $\mathcal{R}_I$ we have the following reduction of the term $t \equiv P(\gamma(\Box), \gamma(\Box), \gamma(\Box))$.

$$\begin{array}{rll} t & \equiv & P(\alpha_{i_1} \ldots \alpha_{i_m}(\Box), \beta_{i_1} \ldots \beta_{i_m}(\Box), \gamma(\Box)) \\ & \to_{\rho_{i_1}} & P(\alpha_{i_2} \ldots \alpha_{i_m}(\Box), \beta_{i_2} \ldots \beta_{i_m}(\Box), \gamma(\Box)) \\ & \to_{\rho_{i_2}} \cdots & \\ & \to_{\rho_{i_m}} & P(\Box, \Box, \gamma(\Box)) \\ & \to_{\rho_{\Box,i}} & P(\gamma(\Box), \gamma(\Box), \gamma(\Box)) \end{array}$$

This is a cyclic reduction, so $\mathcal{R}_I$ is not SN.

($\Leftarrow$) Suppose I has no solution. We now have to show that $\mathcal{R}_I$ is SN. We prove that every term t is SN, with induction on the structure of t. We distinguish cases, according to the head symbol of t.

In the base cases, $t \equiv x$ and $t \equiv \Box$, t is a normal form, so certainly SN.

If $t \equiv 0(t')$ or $t \equiv 1(t')$, then no root steps are possible in a reduction of t, so SN for t follows from the induction hypothesis for t'.

Let $t \equiv P(t_1, t_2, t_3)$. By the induction hypothesis t_1, t_2, t_3 are SN. But then it follows with Exercise 2.3.11 that t_1, t_2, t_3 are also SN with respect to the relation $\rhd$. Now assume t has an infinite reduction sequence. From Observation 5.3.5 (and the Pigeon–Hole Principle) it follows that this reduction sequence must contain a (first) $\rho_{\Box,0}$- or $\rho_{\Box,1}$-step at the root. So the infinite reduction sequence starts with

$$P(t_1, t_2, t_3) \twoheadrightarrow P(\Box, \Box, t') \to P(t', t', t')$$

Repeating this argument we find that

$$P(t', t', t') \twoheadrightarrow P(\Box, \Box, t'') \to P(t'', t'', t'')$$

This is only possible if t' corresponds to a word γ, that, moreover, yields a solution to this instance of PCP, i.e.,

$$\gamma \equiv \alpha_{i_1} \dots \alpha_{i_m} \equiv \beta_{i_1} \dots \beta_{i_m}$$

This contradicts the assumption that I has no solution. ∎

5.4. Undecidability properties of combinatory logic

Without going into much technical detail, we note here that CL has strong undecidability properties. This will be no surprise, given that all partial recursive functions are representable within CL.

The key to the undecidability of CL is the following theorem of Scott [1963]. We state it here without proof. A full account can be found e.g. in Barendregt [1984] or Hankin [1994]. As a matter of fact, the theorem is stated there for the λ-calculus, but because of the equivalence of λ-calculus and CL it carries over immediately.

5.4.1. Theorem (Scott). *Let A be a non-trivial set of ground* CL *terms. By this we mean that A is neither empty, nor the set of all ground terms of* CL. *Suppose, moreover, that A is closed under convertibility, that is, that the implication $t \in A \wedge s = t \;\Rightarrow\; s \in A$ holds for all ground* CL *terms s, t. Then A is undecidable.*

It follows at once that convertibility in CL is undecidable, and the same holds for reduction.

5.4.2. COROLLARY. (i) *The convertibility relation (=) of* CL *is undecidable.*
(ii) *The reduction relation (↠) of* CL *is undecidable.*

PROOF. (i) Assume we have an algorithm that for arbitrary CL terms t, s decides whether $t = s$. Then this algorithm could be used to decide membership of the set $A = \{t \in Ter_0(\mathrm{CL}) \mid t = I\}$. This set is obviously non-empty and closed under convertibility. It is also not the set of all ground CL terms, since by the Church–Rosser theorem for CL we have for example $K \neq I$. So the existence of a decision procedure for A would contradict Scott's theorem.

(ii) The set A we considered in (i) coincides, due to the Church–Rosser property for CL again, with the set $B = \{t \in Ter_0(\mathrm{CL}) \mid t \twoheadrightarrow I\}$. Hence the same reasoning applies. □

It is also undecidable whether a term has a normal form.

5.4.3. PROPOSITION. *Let N be the set*

$$N = \{t \in Ter_0(\mathrm{CL}) \mid \exists n\ (n \text{ is a normal form and } t = n)\}.$$

Then N is undecidable.

PROOF. N is closed under convertibility and neither empty nor the set of all ground CL terms. So Scott's Theorem applies. □

Recall that an S-term in CL is a closed term using only the constant S (Definition 3.2.12). Contrasting with the undecidability results for CL in this section, Waldmann [2000] shows that convertibility for S-terms is decidable. It is also decidable whether an S-term is SN (and hence WN, since for S-terms SN and WN coincide).

5.5. Modularity

A typical organizational concern in building algebraic specifications is their hierarchical or modular structure. The same holds for term rewriting systems. It would be useful if we could prove a property, for example confluence, of such a combined system by just checking it for the separate constituent systems. So it is of obvious interest to determine which properties of term rewriting systems are *modular*, in the sense of carrying over from (smaller) constituent TRSs to a (larger) combined one. Here we think of properties like confluence, termination, uniqueness of normal forms, etc.

As to the way in which TRSs can be combined, the most simple and obvious manner is by taking their union. However, interesting results can be obtained only by imposing some further restrictions. The restriction that is most natural and at the same time the most fruitful is the requirement that the signatures of the component TRSs have to be disjoint. If that is the case we call the union a *disjoint union* or *disjoint sum*.

Historically, the first modularity result was due to Toyama [1987b]: confluence is a modular property. It is an example of a natural but by no means trivial result in the field of term rewriting. In Section 5.6 we will present the simplified proof from Klop et al. [1994]. Toyama's Theorem may be contrasted with the fact that another fundamental property, termination, is not modular. As a matter of fact, this was also discovered by Toyama. Counterexamples will be given in Section 5.7. However, under stricter conditions modularity of termination can yet be obtained. There are several interesting results in this direction. In Section 5.7 we cover the results of Rusinowitch [1987] and Middeldorp [1989], which use restrictions on the distribution between the component TRSs of collapsing and duplicating reduction rules. An elegant treatment of these results will be given, due to Ohlebusch [1993].

One might wonder: if termination is not modular, and confluence is, then maybe their combination, completeness, could be modular again. Example 5.7.2 will show that this is not the case. However, under the further requirement of left-linearity completeness turns out to be modular. This was proved by Toyama, Klop and Barendregt [1989, 1995] and simplified and slightly strengthened by Schmidt-Schauß, Marchiori and Panitz [1995]. The proofs are not included in this book.

5.5.1. Basic notions concerning disjoint sums of TRSs

Term rewriting systems can be combined by taking the union of the signatures and rule sets.

5.5.1. Definition. (i) Consider TRSs $\mathcal{R}_b = (\Sigma_b, R_b)$ and $\mathcal{R}_w = (\Sigma_w, R_w)$. The union TRS $\mathcal{R}_b \cup \mathcal{R}_w$ is defined as $(\Sigma_b \cup \Sigma_w, R_b \cup R_w)$.

(ii) If the signatures of $\mathcal{R}_b, \mathcal{R}_w$ are disjoint, $\Sigma_b \cap \Sigma_w = \varnothing$, then we call $\mathcal{R}_b \cup \mathcal{R}_w$ a *disjoint union* or *disjoint sum*. Notation $\mathcal{R}_b \uplus \mathcal{R}_w$.

5.5.2. Definition. Let P be a property of term rewriting systems. Then P is called *modular* if we have the equivalence: P holds for $\mathcal{R}_b \uplus \mathcal{R}_w$ if and only if P holds for both $\mathcal{R}_b$ and $\mathcal{R}_w$.

By $\to$ we denote reduction in $\mathcal{R}_b \uplus \mathcal{R}_w$. Reduction in $\mathcal{R}_b$ and $\mathcal{R}_w$ separately may be rendered by $\to_b, \to_w$. Note that in general a $\to$-redex in $\mathcal{R}_b \uplus \mathcal{R}_w$ is not either a $\to_b$- or a $\to_w$-redex. The redex can be a *mixed* term, containing symbols from Σ_b as well as from Σ_w.

To facilitate reasoning, we will think of the function and constant symbols in Σ_b as black and those of Σ_w as white. (In print we will sometimes distinguish the 'black' and the 'white' symbols by using capitals versus lower case symbols, or vice versa.) The colour metaphor enables a clear visualization of mixed terms. Thus a term $t \in \mathit{Ter}(\mathcal{R}_b \uplus \mathcal{R}_w)$, in its tree notation, is a constellation of black and white 'triangles', as in Figure 5.2. Formally the triangles are called layers. Call a term *monochrome* if it is either in $\mathit{Ter}(\mathcal{R}_b)$ or in

$Ter(\mathcal{R}_w)$. A *layer* of t can then be characterized as a maximal monochrome subcontext of t. The *top layer* is the maximal monochrome prefix of t. Variables are supposedly colourless. Or rather, a variable always takes the colour of a function symbol directly above it. When discussing an arbitrary term in the sequel we will, without loss of generality, sometimes tacitly assume that its top layer is white.

We already observed that a redex can be a mixed term. However, a redex pattern will always be monochrome.

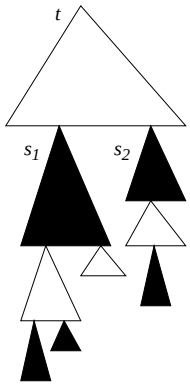


Figure 5.2: Term tree with layer structure.

Going from the top (the root) of a term t, along a path in the term tree, to a terminal symbol one may encounter a number of colour changes and correspondingly traverse a number of layers. The *rank* of a term t is the maximum number of such layers. So a monochrome term has rank 1 and the term in the figure has rank 4. A symbol with a different colour from the one immediately above it is the root of a so-called *special subterm*. Maximal proper special subterms are called *principal subterms*. This is all made precise in the following definition.

5.5.3. Definition. (i) Let $t \equiv C[s_1, \ldots, s_n] \in Ter(\Sigma_b \uplus \Sigma_w)$ and let $C \not\equiv \Box$ be the maximal monochrome prefix of t. That is, C is a Σ_w-context and $root(s_i) \in \Sigma_b$ for $i = 1, \ldots, n$, or the same with colours reversed. Then we write $t \equiv C[\![s_1, \ldots, s_n]\!]$. The occurrences s_i are called the *principal* subterms of t.

(ii) The set $S(t)$ of *special* subterms (more precisely, special subterm occurrences) of t is inductively defined as follows:

$$S(t) = \begin{cases} \{t\} & \text{if } t \in Ter(\mathcal{R}_d) \qquad (d = b, w) \\ \{t\} \cup \bigcup_{1 \leq i \leq n} S(s_i) & \text{if } t \equiv C[\![s_1, \ldots, s_n]\!] \qquad (n > 0) \end{cases}$$

(iii) The *rank* of t is defined inductively as follows:

$$rank(t) = \begin{cases} 1 & \text{if } t \in Ter(\mathcal{R}_d) \qquad (d = b, w) \\ 1 + max(\{rank(s_i) \mid 1 \leq i \leq n\}) & \text{if } t \equiv C[\![s_1, \ldots, s_n]\!] \qquad (n > 0) \end{cases}$$

Note that a non-monochrome term t can be written as $t \equiv C[\![s_1, \ldots, s_n]\!]$ in a unique way. This observation is essential for the above definition to make sense.

5.5.4. EXAMPLE. Consider the term $t \equiv F(g(g(F(a, B))), F(x, g(B)))$ and let $F, B \in \Sigma_w$ and $g, a \in \Sigma_w$. Then t has the white top layer $F(\Box, F(x, \Box))$. The principal subterms of t are $g(g(F(a, B)))$ and $g(B)$. And we have $rank(t) = 4$.

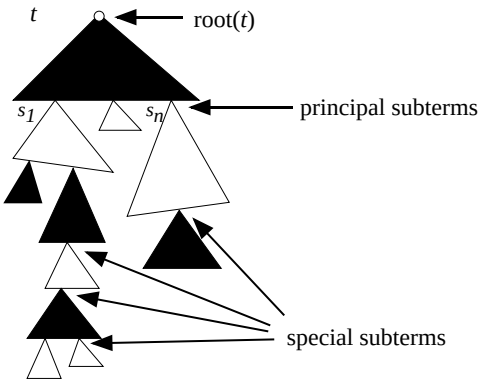


Figure 5.3: Term tree with layer structure: principal and special subterms

Reductions will sometimes change the layer structure. Recall that a reduction rule with a variable as right-hand side is called collapsing (Definition 2.3.3(iv)). A reduction step according to a collapsing reduction rule is also called collapsing. Collapsing steps may change the layer structure, but not necessarily so. We define a stronger notion of a *destructive* step. A destructive reduction step always eliminates a complete layer.

5.5.5. DEFINITION. Let $t \equiv C[r]$, where r is a redex that happens to be also a special subterm of t. Let s be the contractum of r. Then the reduction step $t \to_r C[s]$ is called *destructive* if either s is a variable, or the root symbols of r and s have different colours.

A destructive step will always be collapsing, but of course a collapsing step does not need to be destructive. Since destructive rewrite steps remove a layer from the term tree of t they may decrease the rank. It may be noted, though, that performing a destructive step is not the only way in which the rank can decrease. This can also be the effect of applying an erasing reduction rule.

On the other hand, the rank can never increase under reduction. This fact has a routine proof which is omitted.

5.5.6. PROPOSITION. *If $t \to s$ then $rank(t) \geq rank(s)$.*

We need a distinction between reduction steps that occur inside one of the principal subterms and reduction steps that result by contracting a redex of which the pattern lies in the top layer.

5.5.7. Definition. If $t \to s$ results by contracting a redex that occurs inside one of the principal subterms of t, then we call it an *inner* reduction step, notation $t \to_i s$. Otherwise it is an *outer* step, notation $t \to_o s$. We use the usual notations $\twoheadrightarrow_i$ and $\twoheadrightarrow_o$ for the reflexive–transitive closures of $\to_i$ and $\to_o$.

There are also the corresponding notions of inner and outer redexes. See Figure 5.4. Note that outer reduction can involve destructive steps. So it can very well happen that in a reduction $t \twoheadrightarrow_o s$ the colour of the top layer changes, once or several times.

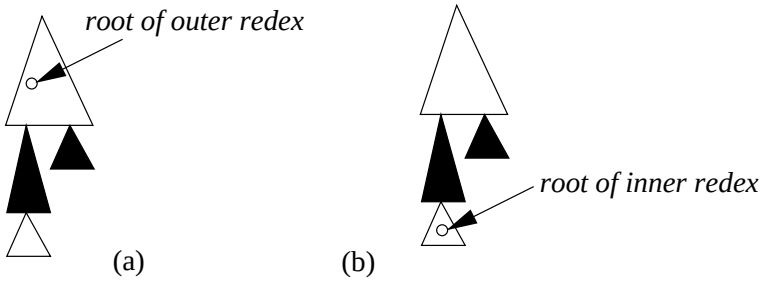


Figure 5.4: Inner and outer redexes

The following notation is useful for dealing with non-left-linear reduction rules.

5.5.8. Notation. We write $\langle s_1, \ldots, s_n \rangle \propto \langle t_1, \ldots, t_n \rangle$ if for all $i, j \leq n$ such that $s_i \equiv s_j$ we have also $t_i \equiv t_j$.

5.5.2. Modularity of weak normalization

We start with a relatively simple observation that is also an instructive case study in proving modularity results.

5.5.9. Proposition. WN *is a modular property.*

Proof. Under the assumption that both $\mathcal{R}_b$ and $\mathcal{R}_w$ are WN, we prove that WN holds for an arbitrary $t \in Ter(\mathcal{R}_b \uplus \mathcal{R}_w)$. We use induction on the rank of t.
Base case. For rank 1 this is the assumption that $\mathcal{R}_i$ is WN, for $i = b, w$.
Inductive case. Suppose every term with rank $\leq n$ is WN. Let t have rank $n+1$. Say the root of t is white. Then $t \equiv C[\![s_1, \ldots, s_m]\!]$, where C is a white context, and the $s_1, \ldots, s_m$ are the principal subterms, with black root and rank $\leq n$.

By the induction hypothesis $s_1, \ldots, s_m$ have normal forms $s_1^\circ, \ldots, s_m^\circ$ (with respect to $\mathcal{R}_b \uplus \mathcal{R}_w$). Now, some of the s_i° may be top white, so let $C[s_1^\circ, \ldots, s_m^\circ] \equiv D[\![u_1, \ldots, u_k]\!]$, with $k \geq 0$. Note that C will be a prefix

of D (still a monochrome white context) and each u_i a subterm of one of the terms $s_1^\circ, \ldots, s_m^\circ$. So the u_i are normal forms.

Choose variables $x_1, \ldots, x_k$ that do not already occur in D. The variables need not be distinct, but the choice should be such that the following condition $(*)$ holds.

$$x_i \equiv x_j \Leftrightarrow u_i \equiv u_j \qquad (*)$$

Then, since $\mathcal{R}_w$ is WN by assumption, the white term $D[x_1, \ldots, x_k]$ reduces to a (white) normal form, say n. So we have $D[x_1, \ldots, x_k] \twoheadrightarrow_w n$.

Now consider the term n° obtained by substituting back the normal forms $u_1, \ldots, u_k$ for the $x_1, \ldots, x_k$ in n. Note that the u_i, of which some may have been copied, some erased, are still the principal subterms of n° – they have black top layers.

We claim that n° is a normal form. A redex cannot occur in any of the u_i, since these are normal forms. Hence a potential redex in n° should have its pattern completely in the white top layer of n°. That pattern would then occur, in the same position, in n as well. Now, for the pattern of a left-linear reduction rule this would obviously contradict the fact that n is a normal form. So the question remains whether, if the pattern belongs to a non-left-linear rule, it could give rise to a redex in n°, but not in n. That could only happen if the substitution of the u_i's for the x_i's would unify two different (white) subterms of n. However, given $(*)$ and the fact that the top layers of the u_i's are black, one easily verifies that that is not possible.

It remains to be pointed out that, indeed, n° is a reduct of t. We have

$$t \equiv C[\![s_1, \ldots, s_m]\!] \twoheadrightarrow C[s_1^\circ, \ldots, s_m^\circ] \equiv D[\![u_1, \ldots, u_k]\!] \twoheadrightarrow n^\circ \qquad \square$$

5.6. Modularity of confluence

In this section we state Toyama's theorem that confluence is a modular property of term rewriting systems. For the proof we refer to Terese [2003], Section 5.8. As a matter of fact, much of the effort in the proof goes into dealing with non-left-linear rewriting rules. Proving modularity of confluence for left-linear TRSs is much simpler. We leave it as an instructive exercise to the reader.

5.6.1. Exercise. Prove that confluence is a modular property of left-linear term rewriting systems.

5.6.2. Theorem (Toyama). *Confluence is a modular property of TRSs.*

The following exercise gives an immediate consequence of Toyama's Theorem.

5.6.3. Exercise. Let $\mathcal{T} = (\Sigma, R)$ be a confluent TRS and let $\Sigma' \supseteq \Sigma$ be any signature extending Σ. Prove that $\mathcal{T}' = (\Sigma', R)$ is confluent.

5.6.4. Exercise. Let $\mathcal{R}_b$ and $\mathcal{R}_w$ be disjoint complete TRSs. Show that every term in $Ter(\mathcal{R}_b \uplus \mathcal{R}_w)$ reduces to a unique normal form.

The following examples were already previewed in Subsection 3.2.4.

5.6.5. Examples. (i) Consider CL $\uplus \{D(x,x) \to E\}$, combinatory logic with binary test for syntactic equality as in Table 3.5. Note that this is indeed a disjoint sum. As we remarked in Section 3.2, CL is a confluent TRS. Trivially, the one-rule TRS $\{D(x,x) \to E\}$ is confluent. Hence, by Toyama's Theorem 5.6.2 the disjoint sum is confluent.

(ii) By contrast, the union CL $\cup \{Dxx \to E\}$, combinatory logic with applicative test for syntactic equality as in Table 3.6, is *not* confluent (Klop [1980]). Note that this combined TRS is not a disjoint sum, since CL and $\{Dxx \to E\}$ have the function symbol for application Ap in common, hidden by the applicative notation.

(iii) Another application of Toyama's Theorem 5.6.2: let $\mathcal{R}$ consist of the rules

$$\begin{array}{lcl} \mathit{if\ true\ then\ } x \mathit{\ else\ } y & \to & x \\ \mathit{if\ false\ then\ } x \mathit{\ else\ } y & \to & y \\ \mathit{if\ } z \mathit{\ then\ } x \mathit{\ else\ } x & \to & x \end{array}$$

(Here $\mathit{true}, \mathit{false}$ are constants and $\mathit{if} - \mathit{then} - \mathit{else} -$ is a ternary function symbol.) Then CL $\uplus\, \mathcal{R}$ is confluent. Again it is essential here that the $\mathit{if} - \mathit{then} - \mathit{else} -$ construct is a ternary operator. If we render $\mathit{if}\, x\, \mathit{then}\, y\, \mathit{else}\, z$ as $Cxyz$, in the applicative notation of CL, with C a constant, the resulting TRS will not be confluent.

5.6.6. Exercise. Prove CR for the extension of combinatory logic

$$\mathrm{CL} \cup \{E(x,x) \to \mathsf{true},\ E(x,S(x)) \to \mathsf{false}\}$$

This fact was already mentioned in Example 4.1.4(ii). See there for further comment.

5.7. The case of strong normalization

In the previous section we proved that confluence is a modular property. One might hope that the same is true for termination (SN), but Toyama [1987b] gives a simple counterexample.

5.7.1. Example. Consider the disjoint TRSs $\mathcal{R}_1, \mathcal{R}_2$ with rules

$$\begin{aligned} R_1 &= \{F(0,1,x) \to F(x,x,x)\} \\ R_2 &= \{or(x,y) \to x,\ or(x,y) \to y\} \end{aligned}$$

Then $\mathcal{R}_1, \mathcal{R}_2$ are both SN (exercise), but $\mathcal{R}_1 \uplus \mathcal{R}_2$ is not, since there is the infinite reduction

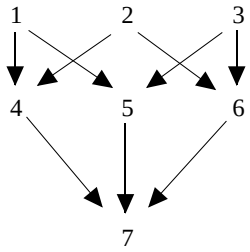
$$\begin{array}{lcl} F(or(0,1), or(0,1), or(0,1)) & \to & F(0, or(0,1), or(0,1)) \\ & \to & F(0,1, or(0,1)) \\ & \to & F(or(0,1), or(0,1), or(0,1)) \\ & \to & \cdots \end{array}$$

In Toyama's counterexample the TRS $\mathcal{R}_2$ is not confluent, and thus one may be tempted to conjecture that the combination of CR and SN, that is, completeness, is a modular property. Again this is not the case, as a counterexample given by Barendregt and Klop (adapted in Toyama [1987a]) shows.

5.7.2. EXAMPLE. $\mathcal{R}_1$ has eleven rules. The first two are

$$\begin{array}{lcl} F(4,5,6,x) & \to & F(x,x,x,x) \\ F(x,y,z,w) & \to & 7 \end{array}$$

The other nine rules of $\mathcal{R}_1$ can be read from the following figure:



The TRS $\mathcal{R}_2$ has the three rules

$$\begin{array}{lcl} g(x,x,y) & \to & x \\ g(x,y,x) & \to & x \\ g(y,x,x) & \to & x \end{array}$$

Now $\mathcal{R}_1$ and $\mathcal{R}_2$ are both complete (exercise), but $\mathcal{R}_1 \uplus \mathcal{R}_2$ is not:

$$\begin{array}{lcccc} & F(g(1,2,3), & g(1,2,3), & g(1,2,3), & g(1,2,3)) \\ \to\!\!\!\to & F(g(4,4,3), & g(5,2,5), & g(1,6,6), & g(1,2,3)) \\ \to\!\!\!\to & F(\quad 4, & \quad 5, & \quad 6, & g(1,2,3)) \\ \to & F(g(1,2,3), & g(1,2,3), & g(1,2,3), & g(1,2,3)) \end{array}$$

A simpler counterexample is given in Drosten [1989]. It is slightly adapted in the following exercise.

5.7.3. EXERCISE. Consider the TRSs $\mathcal{R}_1$, $\mathcal{R}_2$, defined by

$$\begin{array}{lcll} \mathcal{R}_1: & f(0,1,x) & \to & f(x,x,x) \\ & f(x,y,z) & \to & 2 \\ & 0 & \to & 2 \\ & 1 & \to & 2 \end{array}$$

and

$$\begin{array}{lcll} \mathcal{R}_2: & D(x,y,y) & \to & x \\ & D(x,x,y) & \to & y \end{array}$$

(i) Show that $\mathcal{R}_1, \mathcal{R}_2$ are complete.

(ii) Show that the disjoint sum $\mathcal{R}_1 \uplus \mathcal{R}_2$ is not complete. (Hint: Consider the term $f(t,t,t)$ where $t \equiv D(0,1,1)$ and show that $f(t,t,t)$ has a cyclic reduction.)

Observe that each of the two counterexamples we gave involves a non-left-linear TRS. This is essential, as the following theorem indicates.

5.7.4. THEOREM. *Completeness is a modular property for left-linear TRSs.*

This theorem is due to Toyama, Klop and Barendregt [1989, 1995]. The easiest proof is the one in Schmidt-Schauß, Marchiori and Panitz [1995]. It is not included here. We will here only treat the much simpler case where the component TRSs are assumed to be both orthogonal. The proof is a nice illustration of the use of the Erasure Lemma (4.7.4).

5.7.5. THEOREM. SN *is modular for orthogonal TRSs.*

PROOF. First note that from the orthogonality of the component TRSs, it follows that the disjoint sum is also orthogonal, so that in particular the Erasure Lemma applies to the disjoint sum. Consider a minimal counterexample $t \equiv C[\![t_1, \ldots, t_n]\!]$ to the theorem to be proved, minimal with respect to the length of the term (also the rank can be used). So t admits an infinite reduction, but the principal subterms $t_1, \ldots, t_n$ are SN. Now normalize the t_i, result t_i' $(i = 1, \ldots, n)$. We then have $t' \equiv C[\![t_1', \ldots, t_n']\!] \equiv D[s_1, \ldots, s_n]$ with the s_j in normal form. The term $D[x_1, \ldots, x_n]$ is monochrome, hence SN by assumption. One easily sees that then also t' must be SN. But that means that during the reduction $t \twoheadrightarrow t'$ we have 'lost infinity'. According to the Erasure Lemma, this can only happen if a subterm has been erased that admits an infinite reduction. However, the reduction from t to t' took place in the SN terms t_i; so no subterm with infinite reduction could be erased. Contradiction. □

Other useful criteria for the inference of SN for $\mathcal{R}_b \uplus \mathcal{R}_w$ from the SN property for $\mathcal{R}_b$ and $\mathcal{R}_w$ separately can be given in terms of *collapsing* and *duplicating* rewrite rules. These notions were introduced in Definition 2.3.3. The following example serves as a reminder.

5.7.6. EXAMPLE. $F(x,x) \to G(x,x)$ is not a duplicating rule, but $F(x,x) \to H(x,x,x)$ is. Also $P(x) \to G(x,x)$ is a duplicating rule. $F(x, P(y)) \to y$ is a collapsing rule.

5.7.7. THEOREM. *Let $\mathcal{R}_b$ and $\mathcal{R}_w$ be TRSs both with the property SN. Then in each of the following three cases it follows that $\mathcal{R}_b \uplus \mathcal{R}_w$ is SN as well.*

(i) *Neither $\mathcal{R}_b$ nor $\mathcal{R}_w$ contains collapsing rules.*
(ii) *Neither $\mathcal{R}_b$ nor $\mathcal{R}_w$ contains duplicating rules.*
(iii) *One of the TRSs $\mathcal{R}_b, \mathcal{R}_w$ contains neither collapsing nor duplicating rules.*

Part (i) and (ii) of this theorem were first obtained in Rusinowitch [1987]; part (iii) is from Middeldorp [1989]. We will prove the theorem in the different formulation of Theorem 5.7.9. Formulation and proof are both due to

Ohlebusch [1993]. It is straightforward to verify the equivalence between the two versions of the theorem.

5.7.8. Remark. Since there are many cases to consider, it may be convenient to make a table. For $\mathcal{R}_b$ there are 4 cases, according to whether it has only collapsing rules, only duplicating rules, both or neither. For $\mathcal{R}_w$ we have the same, so for the pair $\mathcal{R}_b, \mathcal{R}_w$ there are 16 cases to consider. When we assume $\mathcal{R}_b$ and $\mathcal{R}_w$ to be SN, in 9 of these cases the first formulation of the theorem yields SN for $\mathcal{R}_b \uplus \mathcal{R}_w$. Of the remaining 7 cases there are 2 where Toyama's counterexample yields non-SN for $\mathcal{R}_b \uplus \mathcal{R}_w$. For the remaining 5 cases a simple adaptation of Toyama's counterexample yields the same (by adding some dummy rules to get into the case considered).

5.7.9. Theorem. *Let $\mathcal{R}_b, \mathcal{R}_w$ be disjoint TRSs, both SN, such that $\mathcal{R}_b \uplus \mathcal{R}_w$ is not SN. Then $\mathcal{R}_b$ is duplicating (i.e. contains a duplicating rule) and $\mathcal{R}_w$ is collapsing, or vice versa.*

Proof. Let $\mathcal{R}_b, \mathcal{R}_w$ be two disjoint TRSs, both SN. Let ρ be an infinite reduction $s_0 \to s_1 \to s_2 \to \cdots$ in $\mathcal{R}_b \uplus \mathcal{R}_w$, starting from a term s_0 of minimal rank such that it is not SN. First we note that all s_j have the same rank. This follows from Proposition 5.5.6 in combination with the minimality (with respect to rank) of s_0. A consequence is that the top layer remains of the same colour throughout the reduction. We take it to be white. We claim the following facts to hold:

(i) ρ contains infinitely many outer reduction steps;
(ii) ρ contains infinitely many inner reduction steps which are destructive at level 2;
(iii) ρ contains infinitely many duplicating outer reduction steps.

Here a reduction step is *destructive at level 2* when one of the top layers of a principal subterm is collapsed. See Figure 5.5. Evidently (i), (ii), (iii) imply

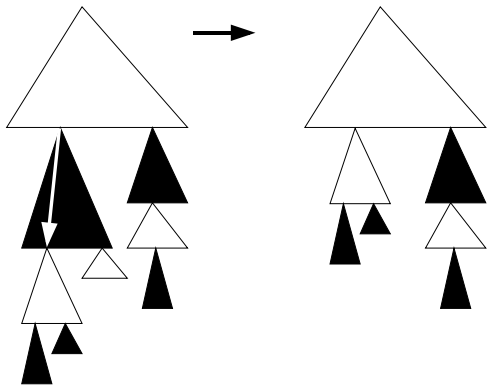


Figure 5.5: A destructive step of level 2.

the theorem. We prove these points successively.

(i) Suppose not. Then, eventually, ρ contains no outer steps. So consider a tail ρ' of ρ, containing no outer steps: $\rho' = s_n \to s_{n+1} \to \cdots$. Then, by a pigeon–hole argument, at least one of the principal subterms of s_n has an infinite reduction. But this principal subterm has a smaller rank than s_n, hence also than s_0. This contradicts the minimality of s_0.

(ii) Suppose not. Again there is a tail $\rho' = s_n \to s_{n+1} \to \cdots$ of ρ which has no inner steps destructive at level 2 at all. Let the root of s_n be in $\mathcal{R}_w$. In the reduction ρ' the (white) top layer is only affected by outer reduction steps, of which there are infinitely many according to (i). Now replace all principal subterms of all the terms s_i by the variable x. Projecting ρ' to these new terms leaves only the outer reduction steps, giving rise to an infinite reduction in $\mathcal{R}_w$. This contradicts the assumption that $\mathcal{R}_w$ is SN.

(iii) Define the norm of t to be the multiset of the ranks of the principal subterms of t (taken as occurrences). It is easy to check that

- only a duplicating outer step increases the norm;
- the norm is preserved or decreasing after a non-duplicating outer step;
- the norm is preserved or decreasing after an inner step;
- the norm decreases after an inner step which is destructive at level 2. (See Figure 5.6.)

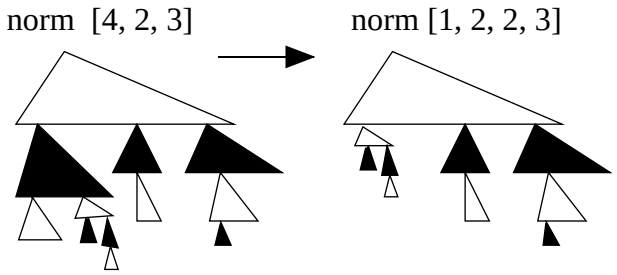


Figure 5.6: Multiset norm decreases after a destructive step at level 2.

By (ii) there are in ρ infinitely many destructive steps of level 2, all decreasing the norm. Hence, by the well-foundedness of the multiset ordering, a norm increase must also happen infinitely many times. This can only be accomplished by duplicating outer steps in ρ, of which there must therefore be infinitely many as well. □

6

Termination

Hans Zantema

This chapter gives an overview of techniques for proving termination of first-order term rewriting systems. Three kinds of techniques are distinguished: semantical methods, syntactical methods and transformational methods. Semantical methods are based on finding a suitable interpretation. They are suitable for all terminating systems but are hard to automate. Syntactical methods like the recursive path order can be automated in a straightforward manner but only apply for a restricted class of term rewriting systems. Therefore a number of transformational methods has been developed by which term rewriting systems outside this class can be transformed to systems inside this class in such a way that termination of the original systems can be concluded.

Nota bene: only thirteen of the original eighty pages of this chapter are included here. For the full chapter see Terese [2003].

6.1. Introduction

In Chapter 5 we saw that termination of finite TRSs is in general undecidable. However, we may expect that there are methods that can be used to establish termination for many TRSs occurring in practical situations. Indeed such methods have been developed, and in this chapter an overview of them is given.

Roughly speaking three kinds of methods can be distinguished: *semantical methods*, *syntactical methods* and *transformational methods*. We briefly discuss these three methods, although in this excerpt only the semantical methods are included. For the other two approaches we refer to the complete version of Terese [2003].

Proving termination by a semantical method means that a weight function has to be defined in such a way that at every reduction step the weight of a term strictly decreases. If the weight is a natural number, or, more generally, a value in a set equipped with a well-founded order, then this cannot go on for ever so termination is ensured. As a framework this is the most basic idea for proving termination; it serves as the basis of a hierarchy of different kinds of termination. About how to choose suitable weight functions only some rough heuristics are available. This approach easily generalizes to termination modulo equations. A special case of this approach is the method of *polynomial interpretations*.

Syntactical methods are based upon orders on terms that are defined by induction on the structure of the terms. Given such an order, if it can be proved that it is well-founded, and if every reduction step causes a decrease with respect to the order, then termination has been proved. Taking the set of terms equipped with such an order makes this approach fit in the general framework of semantical methods. A typical order of this kind is the *recursive path order*. Orders of this kind are usually *precedence-based*: they depend upon an order (called *precedence*) on the function symbols. The advantage of this method is that it provides an algorithm that checks whether termination can be proved by this method or not. This algorithm is easy to implement. The disadvantage is that it covers only a very restricted class of terminating systems. Generalization to termination modulo equations is hardly possible.

Transformational methods provide non-termination preserving transformations between rewriting systems: a transformation Φ falls in this category if termination of a TRS (Σ, R) follows from termination of $\Phi(\Sigma, R)$. If such a Φ can be found in such a way that termination of $\Phi(\Sigma, R)$ can be proved by any other method, then termination of (Σ, R) has been proved. This approach easily generalizes to termination modulo equations.

On each of these kinds of methods a section will be spent. After the treatment of semantical methods a hierarchy of kinds of termination will be given, based on different kinds of well-founded orders in which the weights are taken.

For most methods the notion of order plays an important role. Before the various methods are discussed some basic facts due to Lankford [1979] about the connexion between termination of TRSs and orders are treated. A strict order $<$ is called *well-founded* (Definition A.1.5) if it does not admit an infinite descending sequence $t_0 > t_1 > t_2 > t_3 > \cdots$.

6.1.1. Proposition. *A TRS (Σ, R) is terminating if and only if there exists a well-founded order $<$ on $Ter(\Sigma)$ such that $t > u$ for every $t, u \in Ter(\Sigma)$ for which $t \to_R u$.*

Proof. If (Σ, R) is terminating then choose $<$ to be $\leftarrow_R^+$. On the other hand, if $<$ is an order satisfying the requirements then termination of (Σ, R) follows from well-foundedness of $<$. □

Since usually there are infinitely many pairs of terms $t, u \in Ter(\Sigma)$ for which $t \to_R u$, this proposition is only rarely useful for proving termination of a TRS. In the next proposition we see that with an extra condition on the order it suffices to check $t > u$ only for the rewrite rules instead of for all rewrite steps. Since usually the number of rewrite rules is finite this is much more convenient. First we need a definition.

6.1.2. Definition. A *reduction order* on $Ter(\Sigma)$ is a well-founded order $<$ on $Ter(\Sigma)$ that is

(i) closed under substitutions, i.e., if $t < u$ and σ is an arbitrary substitution then $t^\sigma < u^\sigma$, and

(ii) closed under contexts, i.e., if $t < u$ and C is an arbitrary context then $C[t] < C[u]$.

A reduction order $<$ on $Ter(\Sigma)$ is called *compatible* with a TRS (Σ, R) if $l > r$ for every rewrite rule $l \to r$ in R.

6.1.3. PROPOSITION. *A TRS (Σ, R) is terminating if and only if it admits a compatible reduction order $<$ on $Ter(\Sigma)$.*

PROOF. If (Σ, R) is terminating then again choose $<$ to be $\leftarrow_R^+$. By definition $<$ is closed under substitutions and contexts.

On the other hand, if $<$ is a reduction order that is compatible with (Σ, R), then for an arbitrary rewrite step $t \equiv C[l^\sigma] \to_R C[r^\sigma] \equiv u$ we obtain $t > u$ due to compatibility and closedness under substitutions and contexts. Now termination of (Σ, R) follows from well-foundedness of $<$. □

6.2. Semantical methods

Consider the following two simple examples of TRSs.

$$R_1 \begin{cases} f(f(x)) \to g(x) \\ g(g(x)) \to f(x) \end{cases} \qquad R_2 \begin{cases} f(g(x)) \to f(h(f(x))) \\ g(g(x)) \to f(g(h(x))) \end{cases}$$

Many people immediately see why these TRSs are terminating: R_1 is terminating since at each reduction step the length of the term strictly decreases, and R_2 is terminating since at each reduction step the number of g-symbols strictly decreases. For checking that indeed this kind of weight decreases at every reduction step, it suffices to check that for every rule the weight of the left-hand side is strictly greater than the weight of the right-hand side. In this section this kind of termination proofs is formalized and generalized.

6.2.1. Well-founded monotone algebras

Let Σ be a signature, being a set of operation symbols each having a fixed arity.[1] As in Definition 2.5.1, a Σ-algebra (A, Σ_A) is defined to consist of a non-empty set A, and for every $f \in \Sigma$ a function $f_A : A^n \to A$, where n is the arity of f. This function f_A is called the *interpretation* of f; we write $\Sigma_A = \{f_A \mid f \in \Sigma\}$.

6.2.1. DEFINITION. A *well-founded monotone Σ-algebra* $(A, \Sigma_A, <)$ is a Σ-algebra (A, Σ_A) equipped with a well-founded order $<$ on A such that each algebra operation is strictly monotone in every argument. More precisely, for

[1] Having a fixed arity is not an essential restriction: if a symbol allows more arities, simply replace it by distinct symbols, one for every allowed arity.

every $f \in \Sigma$ and all $a_1, \ldots, a_n, b_1, \ldots, b_n \in A$ with $a_i < b_i$ for some i and $a_j = b_j$ for all $j \neq i$ we have

$$f_A(a_1, \ldots, a_n) < f_A(b_1, \ldots, b_n)$$

Let $(A, \Sigma_A, <)$ be a well-founded monotone Σ-algebra. As before, $Ter(\Sigma)$ denotes the set of terms over Σ and a set Var of variable symbols. Let $\alpha : Var \to A$. We define the term evaluation

$$[\![.]\!]^\alpha : Ter(\Sigma) \to A$$

inductively by

$$\begin{aligned} [\![x]\!]^\alpha &= \alpha(x) \\ [\![f(t_1, \ldots, t_n)]\!]^\alpha &= f_A([\![t_1]\!]^\alpha, \ldots, [\![t_n]\!]^\alpha) \end{aligned}$$

for $x \in Var, f \in \Sigma, t_1, \ldots, t_n \in Ter(\Sigma)$. This function induces a strict partial order $<_A$ on $Ter(\Sigma)$ as follows:

$$t <_A t' \;\Leftrightarrow\; \forall \alpha : Var \to A \; [\![t]\!]^\alpha < [\![t']\!]^\alpha$$

Stated in words, $t <_A t'$ means that for each interpretation of the variables in A the interpreted value of t is smaller than that of t'.

We say that a well-founded monotone algebra $(A, \Sigma_A, <)$ is *compatible* with a TRS if $l >_A r$ for every rule $l \to r$ in the TRS.

Now we arrive at the basic result for this section, which essentially goes back to an unpublished note of Lankford of 1975.

6.2.2. Theorem. *A TRS is terminating if and only if it admits a compatible well-founded monotone algebra.*

In order to give the proof we need two lemmas.

6.2.3. Lemma. *Let $\sigma : Var \to Ter(\Sigma)$ be any substitution and let $\alpha : Var \to A$. Let $\beta : Var \to A$ be defined by $\beta(x) = [\![\sigma(x)]\!]^\alpha$. Then*

$$[\![t^\sigma]\!]^\alpha = [\![t]\!]^\beta$$

for all $t \in Ter(\Sigma)$.

Proof. Induction on the structure of t. This is Exercise 7.1.12(i). □

6.2.4. Lemma. *For any well-founded monotone algebra $(A, \Sigma_A, <)$ the partial order $<_A$ is a reduction order.*

PROOF. Well-foundedness of $<_A$ is immediate from well-foundedness of $<$. It remains to show that $<_A$ is closed under substitutions and contexts. Let $t <_A t'$ for some $t, t' \in Ter(\Sigma)$ and let $\sigma : Var \to Ter(\Sigma)$ be any substitution. Let $\alpha : Var \to A$. From Lemma 6.2.3 we obtain

$$[\![t^\sigma]\!]^\alpha = [\![t]\!]^\beta < [\![t']\!]^\beta = [\![t'^\sigma]\!]^\alpha$$

for some β depending on σ and α. Since this holds for all $\alpha : Var \to A$, we have $t^\sigma <_A t'^\sigma$. Hence $<_A$ is closed under substitutions.

For proving closure under contexts assume $t <_A t'$ for some $t, t' \in Ter(\Sigma)$, and let $f \in \Sigma$. Since $t <_A t'$ we have $[\![t]\!]^\alpha < [\![t']\!]^\alpha$ for all $\alpha : Var \to A$. Applying the monotonicity condition of f_A we obtain

$$\begin{aligned} [\![f(\ldots, t, \ldots)]\!]^\alpha &= f_A(\ldots, [\![t]\!]^\alpha, \ldots) \\ &< f_A(\ldots, [\![t']\!]^\alpha, \ldots) \\ &= [\![f(\ldots, t', \ldots)]\!]^\alpha \end{aligned}$$

This holds for all $\alpha : Var \to A$, so

$$f(\ldots, t, \ldots) <_A f(\ldots, t', \ldots)$$

Closedness under arbitrary contexts follows from this result by induction on the context. □

Now we give the proof of Theorem 6.2.2.

PROOF. If a TRS admits a compatible well-founded monotone algebra $(A, \Sigma_A, <)$ then termination follows from Lemma 6.2.4 and Proposition 6.1.3.

On the other hand, assume the system is terminating. Define $A = Ter(\Sigma)$, let $f_A(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$, and define $<$ to be the transitive closure of the inverse of the rewrite relation. One easily verifies that $(A, \Sigma_A, <)$ is a well-founded monotone algebra. We still have to prove that $l >_A r$ for each rewrite rule $l \to r$. Let $\alpha : Var \to A$. Since $A = Ter(\Sigma)$ we see that α is a substitution. Then $[\![t]\!]^\alpha = t^\alpha$ for each term t, which is easily proved by induction on the structure of t. Since $l \to r$ is a rewrite rule, the term l^α can be reduced in one step to r^α. So

$$[\![l]\!]^\alpha \;=\; l^\alpha \;>\; r^\alpha \;=\; [\![r]\!]^\alpha$$

This holds for every $\alpha : Var \to A$, so $l >_A r$. □

A way of proving termination of a TRS is now as follows: choose a set A equipped with a well-founded order $<$, define for each operation symbol f a corresponding operation f_A that is strictly monotone in all of its arguments, and for which $[\![l]\!]^\alpha > [\![r]\!]^\alpha$ for all rewrite rules $l \to r$ and all $\alpha : Var \to A$. Then according to Theorem 6.2.2 the TRS is terminating.

The problem is how to choose the partially ordered set and the operations. The simplest useful choice for $(A, <)$ is $(\mathbb{N}^+, <)$, the set of strictly positive integers with the ordinary order. In many applications this is already a fruitful choice.

6.2.5. Example. The two simple examples at the beginning of this section can be seen as applications of Theorem 6.2.2 with this choice for $(A, <)$ as follows. The length of terms over f, g is modelled by choosing $f_A(x) = g_A(x) = x + 1$ for all $x \in A$. Indeed f_A and g_A are both strictly monotone, and

$$f_A(f_A(x)) = x + 2 > x + 1 = g_A(x) \quad \text{and} \quad g_A(g_A(x)) = x + 2 > x + 1 = f_A(x)$$

for all $x \in A$. Hence $f(f(x)) >_A g(x)$ and $g(g(x)) >_A f(x)$, proving termination of R_1 by Theorem 6.2.2.

The number of gs in terms over f, g, h is modelled by choosing $f_A(x) = h_A(x) = x$ and $g_A(x) = x + 1$ for all $x \in A$. Clearly f_A, g_A and h_A are all strictly monotone, and

$$\begin{array}{lcccccl} f_A(g_A(x)) & = & x+1 & > & x & = & f_A(h_A(f_A(x))) \quad \text{and} \\ g_A(g_A(x)) & = & x+2 & > & x+1 & = & f_A(g_A(h_A(x))) \end{array}$$

for all $x \in A$, proving termination of R_2 by Theorem 6.2.2.

6.2.6. Example. As another application of the choice $(A, <) = (\mathbb{N}^+, <)$ we consider associativity. Consider the system consisting of one single rule

$$f(f(x, y), z) \;\to\; f(x, f(y, z))$$

and choose $f_A(x, y) = 2x + y$ for all $x, y \in A$. Clearly f_A is strictly monotone in both arguments, and

$$f_A(f_A(x, y), z) = 4x + 2y + z > 2x + 2y + z = f_A(x, f_A(y, z))$$

for all $x, y, z \in A$, proving termination according to Theorem 6.2.2.

6.2.7. Exercise. Prove termination of the following TRSs, each consisting of a single rule, by choosing $(A, <) = (\mathbb{N}^+, <)$.

(i) $f(g(x)) \to g(f(x))$;
(ii) $f(g(x)) \to g(g(f(x)))$;
(iii) $f(g(x)) \to a(g(f(x)), x)$.

6.2.2. Polynomial interpretations

In the basic applications of Theorem 6.2.2 the well-founded monotone algebra $(A, \Sigma_A, <)$ consists of the natural numbers with the usual order, and for all $f \in \Sigma$ the function f_A is a polynomial in its arguments. For this kind of polynomial interpretations some heuristics have been developed to choose the polynomials, and some techniques to prove that $l >_A r$ for rewrite rules $l \to r$ (Ben-Cherifa and Lescanne [1987]). Here we give an overview of these techniques. We start with a number of definitions.

6.2.8. Definition. A *monomial* in n variables over $\mathbb{Z}$ is a function $F : \mathbb{Z}^n \to \mathbb{Z}$ defined by $F(x_1, \ldots, x_n) = ax_1^{k_1}x_2^{k_2}\ldots x_n^{k_n}$ for some integer $a \neq 0$ and some non-negative integers $k_1, k_2, \ldots, k_n$. The number a is called the *coefficient* of the monomial. If $k_1 = k_2 = \cdots = k_n = 0$ then the monomial is called a *constant*.

For functions $F_i : \mathbb{Z}^n \to \mathbb{Z}$, $i = 1, \ldots, m$, the *sum* and the *product* are defined respectively as follows:

$$\Big(\sum_{i=1}^{m} F_i\Big)(x_1, \ldots, x_n) = \sum_{i=1}^{m} F_i(x_1, \ldots, x_n)$$

$$\Big(\prod_{i=1}^{m} F_i\Big)(x_1, \ldots, x_n) = \prod_{i=1}^{m} F_i(x_1, \ldots, x_n)$$

A *polynomial* in n variables over $\mathbb{Z}$ is the sum of finitely many monomials in n variables over $\mathbb{Z}$.

Choose

$$A = \{n \in \mathbb{N} \mid n \geq 2\}$$

A *polynomial interpretation* for a signature Σ consists of a polynomial f_A in n variables for every symbol $f \in \Sigma$, where n is the arity of f, such that for all $f \in \Sigma$

(i) $f_A(x_1, \ldots, x_n) \in A$ for all $x_1, \ldots, x_n \in A$ (*well-definedness*), and
(ii) f_A is strictly monotone in all of its arguments.

Now $(A, \Sigma_A, <)$ is a well-founded monotone algebra and the order $<_A$ is defined as before.

A polynomial interpretation $\{f_A \mid f \in \Sigma\}$ is *compatible* with a TRS (Σ, R) if $l >_A r$ for every rule $l \to r$ in R, or, stated equivalently, if the well-founded monotone algebra $(A, \Sigma_A, <)$ is compatible with (Σ, R).

A TRS is *polynomially terminating* if it admits a compatible polynomial interpretation.

Usually the monomial F defined by $F(x_1, \ldots, x_n) = x_i$ is denoted by X_i, for $i = 1, \ldots, n$; if $n \leq 3$ one often writes $X = X_1, Y = X_2, Z = X_3$. A constant monomial is denoted by its coefficient. In this way the monomial $F : \mathbb{Z}^n \to \mathbb{Z}$ defined by $F(x_1, \ldots, x_n) = ax_1^{k_1}x_2^{k_2}\ldots x_n^{k_n}$ for an integer $a \neq 0$ and non-negative integers $k_1, k_2, \ldots, k_n$ is written as $aX_1^{k_1}X_2^{k_2}\ldots X_n^{k_n}$.

One easily sees that

(i) the product of finitely many monomials is again a monomial,
(ii) the sum of finitely many polynomials is again a polynomial,
(iii) the product of finitely many polynomials is again a polynomial.

For instance, $3XZ + Y^3$ is a polynomial.

Due to Theorem 6.2.2 polynomial termination indeed implies termination. We like to stress here that strict monotonicity in all arguments is essential

for this statement. For instance, the rule $f(s(x), y) \to f(x, f(s(x), y))$ is not terminating although $f(s(x), y) >_A f(x, f(s(x), y))$ holds in the interpretation $f_A = X, s_A = X + 1$. This interpretation does not fulfil the requirements of Theorem 6.2.2: f_A is not strictly monotone in its second argument.

The reason for choosing $A = \{n \in \mathbb{N} \mid n \geq 2\}$ instead of $A = \mathbb{N}$ is merely for convenience. For the power of the interpretations this does not have any effect since this set is order-isomorphic to $\mathbb{N}$ by an order-isomorphism adding 2 to every element, and the composition of polynomials and such isomorphisms again yields polynomials. But this choice has a few advantages over the natural numbers, for instance excluding zero yields that multiplication is strictly monotone in both arguments, and excluding 1 yields that the polynomial XY pointwise exceeds the polynomials X and Y. As a drawback of this choice for a constant we may not choose a value less than 2.

The following proposition gives sufficient (but not necessary) conditions for well-definedness and strict monotonicity.

6.2.9. Proposition. *Let $F = \sum_{i=1}^{m} a_i X_1^{k_{i,1}} X_2^{k_{i,2}} \dots X_n^{k_{i,n}}$ be a polynomial in $n > 0$ variables for which $a_i > 0$ for $i = 1, \dots, m$, and for every $j = 1, \dots, n$ there exists i with $1 \leq i \leq m$ and $k_{i,j} > 0$. Then F is well-defined, and F is strictly monotone in all its arguments.*

Proof. Every non-constant monomial with coefficient ≥ 1 yields a value ≥ 2 for every interpretation of the variables in A. Since $n > 0$ there exists $k_{i,j} > 0$, hence at least one of the monomials is non-constant. Hence F is well-defined.

A monomial $aX_1^{k_1} X_2^{k_2} \dots X_n^{k_n}$ with $a > 0$ is strictly monotone in the ith argument if $k_i > 0$, and is weakly monotone in all other arguments. Hence F is strictly monotone in all its arguments. □

The condition in Proposition 6.2.9 that every variable occurs in at least one of the monomials cannot be weakened: if some variable does not occur in any of the monomials then the polynomial is not strictly monotone in the corresponding argument. For instance, the polynomial $Y + YZ^2$ in three variables X, Y, Z is not strictly monotone in the first argument.

The condition in Proposition 6.2.9 that all coefficients are positive is not necessary: for instance $X^2 - X$ is a well-defined and strictly monotone polynomial in one variable. However, in most applications negative coefficients do not occur and Proposition 6.2.9 can be applied to establish well-definedness and strict monotonicity.

For a rewrite rule $l \to r$ let $x_1, \dots, x_n$ be the variables occurring in l and r. Given a polynomial interpretation define $F_{l,r} : A^n \to \mathbb{Z}$ by

$$F_{l,r}(a_1, \dots, a_n) = [\![l]\!]^\alpha - [\![r]\!]^\alpha$$

for α defined by $\alpha(x_i) = a_i$ for $i = 1, \dots, n$. One easily checks that $F_{l,r}$ is a polynomial in n variables; often some coefficients will be negative. By

definition $l >_A r$ if and only if $F_{l,r}(a_1, \ldots, a_n) > 0$ for all $a_1, \ldots, a_n \in A$. A polynomial F is called *strictly positive* if $F(a_1, \ldots, a_n) > 0$ for all $a_1, \ldots, a_n \in A$.

In order to prove termination of a TRS (Σ, R) by means of a polynomial interpretation we have to choose a polynomial f_A in n variables for every symbol $f \in \Sigma$, where n is the arity of f, satisfying the following three conditions.

(i) The polynomial f_A is well-defined. For a constant this means that we have to choose a value ≥ 2; if the arity of f is positive and f_A is of the required form this is obtained for free by Proposition 6.2.9.

(ii) The polynomial f_A is strictly monotone in all of its arguments. For a constant this condition is empty; if the arity of f is positive and f_A is of the required form this is obtained for free by Proposition 6.2.9.

(iii) The interpretation has to be compatible, that is, $F_{l,r}(a_1, \ldots, a_n) > 0$ for all $a_1, \ldots, a_n \in A$ and every rule $l \to r$.

We conclude that the method now consists of two main steps:

(i) Choose suitable polynomials with positive coefficients for all symbols $f \in \Sigma$ in which all variables occur.

(ii) Prove that $F_{l,r}(a_1, \ldots, a_n) > 0$ for all $a_1, \ldots, a_n \in A$ for the corresponding polynomials $F_{l,r}$ for all rules $l \to r$ of R.

Regarding the first step only some rough heuristics can be given. After giving a number of examples we will summarize a few. Often the first choice is not yet successful and some backtracking is applied for finding the right choice.

We conclude this section by some examples and some heuristics for choosing the polynomials.

6.2.10. Example.

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

This system describes the usual arithmetic on natural numbers. Choose $0_A = 2$, $s_A = X + 3$, $+_A = 2X + Y$, $*_A = XY$. Then the polynomials $F_{l,r}$ for the four rules are 4, 3, $2X - 2$, Y, respectively, all of which are clearly strictly positive, proving polynomial termination.

6.2.11. Example.

$$\begin{array}{rcl} x + x & \to & x \\ (x + y) \cdot z & \to & (x \cdot z) + (y \cdot z) \\ (x \cdot y) \cdot z & \to & x \cdot (y \cdot z) \\ x + \delta & \to & x \\ \delta \cdot x & \to & \delta \end{array}$$

This system describes part of a process algebra where $+$ denotes choice, $\cdot$ denotes sequential composition and δ denotes deadlock. Choose $\delta_A = 2$, $+_A = X+Y$, $\cdot_A = X^2Y$. Now the polynomials $F_{l,r}$ for the five rules are X, $2XYZ$, $X^4Y^2Z - X^2Y^2Z$, 2, $4X-2$, respectively, all of which are checked to be strictly positive automatically by the above method, proving polynomial termination.

As heuristics for choosing the interpretations we mention:

(i) Choose $c_A = 2$ for constants c.

(ii) Concentrate on rules for which the right-hand side looks more complicated than the left-hand side.

(iii) If the operations have some arithmetical meaning, try to choose an interpretation that slightly exceeds the arithmetical meaning.

(iv) If the operations have a natural precedence, try to choose polynomials with a low degree for the lowest symbols and polynomials with a higher degree for higher symbols.

(v) If a left-hand side of a rule $l \to r$ is of the form $f(x_1, \ldots, x_n)$ then try to choose $f_A(a_1, \ldots, a_n) = [\![r]\!]^\alpha + 1$ for α defined by $\alpha(x_i) = a_i$ for $i = 1, \ldots, n$.

(vi) For an associative rule $x + (y + z) \to (x + y) + z$ try to choose $+_A$ to be one of the polynomials $X + 2Y, XY^2, X + Y^2, XY + Y$; for the reversed associative rule interchange X and Y.

(vii) For a distributive rule $x * (y + z) \to (x * y) + (x * z)$ or an endomorphism rule $f(x + y) \to f(x) + f(y)$ try to choose either $+_A = aX + bY$ for some $a, b \geq 1$, $*_A = XY^2, f_A = X^2$, or $+_A = aX + bY + c$ for some $a, b, c \geq 1$, $*_A = XY, f_A = 2X$.

6.2.12. Exercise. Prove termination of the following TRSs by means of polynomials.

(i) $f(g(x)) \to h(x), \quad h(x) \to g(f(x))$

(ii)
$$\begin{cases} 0 + x & \to & x \\ x + 0 & \to & x \\ x * 0 & \to & 0 \\ x * (y + z) & \to & (x * z) + (y * x) \end{cases}$$

6.2.3. Lexicographic combinations

Sometimes, while trying to prove polynomial termination, one achieves interpretations for which the left-hand side is not strictly greater than the right-hand side, but only greater or equal. This can still be fruitful in proving termination: if the smaller TRS consisting of the rules for which this non-strict inequality holds is terminating, then the whole system can be concluded to be terminating as well. In the general framework of monotone algebras this is stated in the next proposition; for a well-founded monotone

algebra $(A, \Sigma_A, <)$ write

$$t \geq_A u \quad \Leftrightarrow \quad \forall \alpha : \mathit{Var} \to A \; [\![t]\!]^\alpha \geq [\![u]\!]^\alpha$$

Note that $\geq_A$ is essentially more than the union of $>_A$ and equality.

6.2.13. Proposition. *Let (Σ, R) be a TRS for which $R = R' \cup R''$ and let $(A, \Sigma_A, <)$ be a well-founded monotone Σ-algebra for which*

(i) *$l >_A r$ for every rule $l \to r$ in R', and*
(ii) *$l \geq_A r$ for every rule $l \to r$ in R'', and*
(iii) *(Σ, R'') is terminating.*

Then (Σ, R) is terminating.

Proof. Due to Theorem 6.2.2 R'' admits a compatible well-founded monotone algebra $(B, \Sigma_B, \prec)$. Now define $C = A \times B$ with the lexicographic order

$$(a, b) \ll (a', b') \quad \Leftrightarrow \quad a < a' \vee (a = a' \wedge b \prec b')$$

(see Definition A.1.15) and for every $f \in \Sigma$

$$f_C((a_1, b_1), \ldots, (a_n, b_n)) = (f_A(a_1, \ldots, a_n), f_B(b_1, \ldots, b_n))$$

for $a_1, \ldots, a_n \in A, b_1, \ldots, b_n \in B$. For $\gamma : \mathit{Var} \to C$ we can write $\gamma(x) = (\alpha(x), \beta(x))$ for every $x \in \mathit{Var}$, for some $\alpha : \mathit{Var} \to A$, $\beta : \mathit{Var} \to B$. One easily checks

$$[\![t]\!]^\gamma = ([\![t]\!]^\alpha, [\![t]\!]^\beta)$$

for every term t, and hence the well-founded monotone algebra $(C, \Sigma_C, \ll)$ is compatible with R. Due to Theorem 6.2.2 now (Σ, R) is terminating. □

6.2.14. Example. The following TRS is closely related to the one from Example 6.2.10:

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

However, the choice $0_A = 2, s_A = X + 3, +_A = 2X + Y, *_A = XY$ now fails for the last rule. It can even be shown that this system is not polynomially terminating: no other choice of polynomials is possible.[2] A termination proof can be given by using elementary functions instead of polynomials: choose $0_A = 2, +_A = 2X + Y, s_A = X + 2$ and $x *_A y = y * 2^x$.

[2] A rough sketch of the proof is the following. Write $x +_A y = x * f(x, y) + g(y)$. Taking x constant in the last rule yields $p(y) > q(y) * f(q(y), y) + g(y)$ for two polynomials p, q of the same degree. Considering degrees in this inequality yields that f is a constant. Then the second rule yields that $f > 1$ and $s_A = X + c$ for some constant c. Again take x constant in the last rule, then considering the leading coefficient gives rise to a contradiction.

However, using a lexicographic combination there is no reason to leave polynomials: choose $0_A = 2, +_A = X + Y, s_A = X + 2, *_A = XY$, then we get equality for the second rule and strict inequality for the other rules. Hence we can apply Proposition 6.2.13 by choosing R'' to consist of the second rule and R' to consist of the rest. Termination of R'' follows from the interpretation $+_A = 2X + Y, s_A = X + 1$, hence termination of the full system follows from Proposition 6.2.13.

Clearly Proposition 6.2.13 easily extends to n-tuples of TRSs instead of only pairs (R', R'').

6.2.4. Other examples

We conclude this section with three examples of termination proofs by means of well-founded monotone algebras in which the orders are not total. For all three cases we define

$$A = \{0, 1\} \times \mathbb{N}^+ \quad \text{and} \quad (a, n) < (b, m) \quad \Leftrightarrow \quad a = b \wedge n < m$$

Note that $<$ is indeed a well-founded partial order on A which is not total. The overloading of the symbol $<$ does not cause confusion since from the form of the arguments it is clear which of the two orders is intended.

6.2.15. Example. Consider the TRS consisting of the rule

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

The intuition is that if f is applied to $f(\ldots)$ we want to obtain a higher weight than if f is applied to $g(\ldots)$. To be able to distinguish between these arguments $f(\ldots)$ and $g(\ldots)$ we let $f_A(\ldots)$ always result in $(1, \ldots)$ and we let $g_A(\ldots)$ always result in $(0, \ldots)$. Define

$$f_A(0, n) = (1, n), \quad f_A(1, n) = (1, n + 1), \quad g_A(0, n) = (0, n), \quad g_A(1, n) = (0, n)$$

for all $n \in \mathbb{N}$. Both f_A and g_A are strictly monotone, and

$$\begin{array}{lcllcl} f_A(f_A(0,n)) & = & (1, n+1) & > & (1, n) & = & f_A(g_A(f_A(0,n))) \\ f_A(f_A(1,n)) & = & (1, n+2) & > & (1, n+1) & = & f_A(g_A(f_A(1,n))) \end{array}$$

for all $n \in \mathbb{N}$, proving termination.

6.2.16. Example. Consider the TRS with the two rules

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

Define

$$f_A(0, n) = (1, 2n), \quad f_A(1, n) = (1, n + 1)$$

$$g_A(0, n) = (0, n + 1), \quad g_A(1, n) = (0, 2n)$$

Both f_A and g_A are strictly monotone, while

$$\begin{array}{lcllcl} f_A(g_A(0,n)) & = & (1,2n+2) & > & (1,2n+1) & = & f_A(f_A(0,n)) \\ f_A(g_A(1,n)) & = & (1,4n) & > & (1,n+2) & = & f_A(f_A(1,n)) \\ g_A(f_A(0,n)) & = & (0,4n) & > & (0,n+2) & = & g_A(g_A(0,n)) \\ g_A(f_A(1,n)) & = & (0,2n+2) & > & (0,2n+1) & = & g_A(g_A(1,n)) \end{array}$$

for all $n \in \mathbb{N}$, proving termination.

6.2.17. Example. Let the TRS consist of the rule

$$f(0,1,x) \;\to\; f(x,x,x)$$

The origin of this TRS is the example of Toyama [1987a] showing that termination is not modular (Example 5.7.1. After that it has served as a counterexample for many properties. Define

$$0_A = (0,1), \quad 1_A = (1,1)$$

$$f_A((a,n),(b,m),(c,k)) = \begin{cases} (0,n+m+k) & \text{if } a = b \\ (0,n+m+3k) & \text{if } a \neq b \end{cases}$$

The function f_A is strictly monotone in all three arguments. For all $(a,n) \in A$ we have

$$f_A(0_A,1_A,(a,n)) = (0,3n+2) > (0,3n) = f_A((a,n),(a,n),(a,n))$$

proving termination.

6.2.18. Exercise. Prove that the following single-rule TRSs are both terminating.

(i) $f(g(x)) \to f(s(s(g(x))))$;

(ii) $f(0,x) \to f(s(0),x)$.

7

Completion of equational specifications

Inge Bethke

This chapter examines rewriting techniques as tools for the validity problem of equational specifications. The first two sections develop the basic theory of equational deduction and initial algebra semantics. Then we embark on the study of completion procedures in practical and abstract settings.

Nota bene: this is about half of the original chapter in Terese [2003].

Equational specification techniques have gained considerable importance in the whole area of computer science, and are now playing a central role in the theory of abstract data type specification. In this chapter, we will give an introduction to the area of Knuth–Bendix completion which – given an equational specification – is concerned with the construction of a complete TRS which proves the same equations as the original specification in the sense that convertibility and provable equality coincide. If the construction is successful, it yields a positive solution to the validity problem as already mentioned in Section 2.4.

7.1. Equational specifications: syntax and semantics

There are close connexions between TRSs and equational specifications. One obvious connexion is that equational specifications can be seen as TRSs "without orientation".

7.1.1. Definition. Let Σ be a signature.

(i) An *equational specification* is a pair (Σ, E) where $E \subseteq Ter(\Sigma)^2$ is a set of "equations". Instead of (s, t) we will often write equations as $s = t$,tacitly assuming that there cannot arise any confusion with convertibility.

(ii) An equational specification (Σ, E) defines an equational theory consisting of all equations derivable from E by means of *equational logic*. The axioms and rules of equational logic are the ones listed in Table 7.1. If an equation $s = t$ is derivable from the equations in E by means of equational logic, we write $(\Sigma, E) \vdash s = t$.

Axioms :	$s = t$	(E)	for all $(s,t) \in E$
	$t = t$	(ref)	for every $t \in Ter(\Sigma)$
Rules :	$\dfrac{s = t}{t = s}$	(sym)	
	$\dfrac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$	$(trans)$	
	$\dfrac{s = t}{s^\sigma = t^\sigma}$	(sub)	for every substitution σ
	$\dfrac{s_1 = t_1 \ \cdots \ s_n = t_n}{F(s_1,\ldots,s_n) = F(t_1,\ldots,t_n)}$	(F)	for every n-ary $F \in \Sigma$

Table 7.1: Axioms and rules of the equational logic E

7.1.2. EXAMPLE. Consider the simple specification (Σ, E) of the natural numbers with addition where $\Sigma = \{0, S, Add\}$ and E consists of the two equations

$$\begin{array}{rcl} Add(x,0) & = & x \\ Add(x,S(y)) & = & S(Add(x,y)) \end{array}$$

Then $(\Sigma, E) \vdash Add(S(0),S(0)) = S(S(0))$:

$$\dfrac{\dfrac{Add(x,S(y)) = S(Add(x,y)) \ \ (E)}{Add(S(0),S(0)) = S(Add(S(0),0))}\,(sub) \quad \dfrac{\dfrac{Add(x,0) = x \ \ (E)}{Add(S(0),0) = S(0)}\,(sub)}{S(Add(S(0),0)) = S(S(0))}\,(S)}{Add(S(0),S(0)) = S(S(0))}\,(trans)$$

7.1.3. EXERCISE. Recall the traverses from a TRS to a corresponding equational specification and back given in Definition 2.4.2: For a pseudo-TRS (Σ, R), the corresponding equational specification is the specification $(\Sigma, R^=)$, where the set of equations is defined by $R^= = \{s = t \mid s \to t \in R\}$. Conversely, for an equational specification (Σ, E), the corresponding pseudo-TRS is the pseudo-TRS $(\Sigma, E^\leftrightarrow)$, where the set of reduction rules is defined by $E^\leftrightarrow = \{s \to t \mid s = t \in E \text{ or } t = s \in E\}$. We write $s =_E t$ if s and t are convertible in $E^\leftrightarrow$.

Now let (Σ, R) be a TRS. Prove that $(\Sigma, R^=) \vdash s = t$ if and only if $s =_R t$, and conclude that, for every equational specification (Σ, E), we have $(\Sigma, E) \vdash s = t$ if and only if $s =_E t$.

7.1.4. EXERCISE. Often the inference system in Table 7.1 is presented in a slightly different version where the rules (sub) and (F) are replaced by the rules

$$\dfrac{s_1 = s_2}{s_1[x := t] = s_2[x := t]} \quad (sub1) \quad \text{for every } x, t \in Ter(\Sigma)$$

$$\dfrac{s_1 = s_2}{t[x := s_1] = t[x := s_2]} \quad (sub2) \quad \text{for every } x, t \in Ter(\Sigma)$$

A third formulation is obtained by combining the two substitution rules of that system into one, i.e. by replacing the rules ($sub1$) and ($sub2$) by

$$\frac{s_1 = s_2 \quad t_1 = t_2}{s_1[x := t_1] = s_2[x := t_2]} \quad (sub1, 2) \quad \text{for every variable } x$$

Prove the equivalence of these versions with the system in Table 7.1.

An equational specification comes along with a natural semantics obtained by giving some meaning to the symbols of its signature Σ. If we give these meanings, we get a Σ-algebra as in Definition 2.5.1. Probably the best-known $\{0, S, Add\}$-algebra is the algebra $\mathcal{N}$ of natural numbers with $0^{\mathcal{N}} = 0$, and $S^{\mathcal{N}}(n) = n + 1$ and $Add^{\mathcal{N}}(n, m) = n + m$ for $n, m \in \mathbb{N}$. In addition to this natural algebra there exist for any signature two term algebras which are of a particular technical interest. For the signature $\{0, S, Add\}$ the elements of the *ground term algebra* are 0, $S(0)$, $Add(0, 0)$, $Add(0, S(0))$ etc. What are the functions? What should e.g. Add do when applied to 0 and $Add(0, S(0))$? It should produce $Add(0, Add(0, S(0)))$ of course. The functions simply build larger terms from smaller ones. A slightly more general concept is the *term algebra*. The set of elements of this algebra is the set of all terms, not only ground terms but also terms containing variables. For example, as well as $Add(0, 0)$ we have $Add(x, 0)$. The functions are the term constructors like Add, S as before.

7.1.5. Definition. Let Σ be a signature. The *term algebra*, simply denoted by $Ter(\Sigma)$, consists of $Ter(\Sigma)$ together with the functions $F^{Ter(\Sigma)}$ defined by

$$F^{Ter(\Sigma)}(t_1, \ldots, t_n) = F(t_1, \ldots, t_n)$$

for every n-ary $F \in \Sigma$. Similarly, we define the *ground term algebra* with $Ter_0(\Sigma)$ instead of $Ter(\Sigma)$.

7.1.6. Remark. $Ter_0(\Sigma)$ only qualifies as a Σ-algebra if $Ter_0(\Sigma) \neq \varnothing$, i.e. if Σ contains at least one constant. This can of course always be achieved by augmenting Σ by a dummy constant without, in the case of entire specifications, modifying the set of equations. We shall henceforth leave signatures inducing an empty set of ground terms outside consideration.

An important Σ-algebra construction is the quotient construction associated with an equivalence relation on a Σ-algebra.

7.1.7. Definition. Let Σ be a signature and $\mathcal{A}$ be a Σ-algebra. A Σ-*congruence* on $\mathcal{A}$ is an equivalence relation $\sim$ on the carrier set A which is compatible with the functions on A, i.e. which satisfies the following condition: for each n-ary function symbol $F \in \Sigma$ and any $a_1, \ldots, a_n, a'_1, \ldots, a'_n \in A$, if $a_i \sim a'_i$ for $i = 1, \ldots, n$, then $F^{\mathcal{A}}(a_1, \ldots, a_n) \sim F^{\mathcal{A}}(a'_1, \ldots, a'_n)$.

7.1.8. Exercise. Let (Σ, E) be an equational specification. Show that the relation $=_E$ is a Σ-congruence on $Ter(\Sigma)$ and $Ter_0(\Sigma)$.

7.1.9. Definition. Let $\mathcal{A}$ be a Σ-algebra with carrier set A and let $\sim$ be a Σ-congruence on $\mathcal{A}$. The *quotient algebra* $\mathcal{A}/\sim$ is the algebra consisting of the quotient modulo $\sim$ $A/\sim$ (see A.1.5) together with the functions $F^{\mathcal{A}/\sim}$ defined by

$$F^{\mathcal{A}/\sim}([a_1]_\sim, \ldots, [a_n]_\sim) \;=\; [F^{\mathcal{A}}(a_1, \ldots, a_n)]_\sim$$

for every n-ary $F \in \Sigma$.

7.1.10. Lemma. *Let $\mathcal{A}$ be a Σ-algebra and $\sim$ be a Σ-congruence on $\mathcal{A}$. Then $\mathcal{A}/\sim$ is a Σ-algebra.*

Proof. We have to show that $F^{\mathcal{A}/\sim}$ is well-defined, i.e. does not depend on the choice of the representatives for equivalence classes. Thus suppose $a_i \sim a_i'$ for each $0 \leq i \leq n$. Then $F^{\mathcal{A}}(a_1, \ldots, a_n) \sim F^{\mathcal{A}}(a_1', \ldots, a_n')$, since $\sim$ is compatible with $F^{\mathcal{A}}$. Hence $[F^{\mathcal{A}}(a_1, \ldots, a_n)]_\sim = [F^{\mathcal{A}}(a_1', \ldots, a_n')]_\sim$ and therefore $F^{\mathcal{A}/\sim}([a_1]_\sim, \ldots, [a_n]_\sim) = F^{\mathcal{A}/\sim}([a_1']_\sim, \ldots, [a_n']_\sim)$. □

7.1.11. Notation. Note that for every equational specification (Σ, E), both $Ter(\Sigma)/=_E$ and $Ter_0(\Sigma)/=_E$ are Σ-algebras by the preceding lemma and Exercises 7.1.8 and 7.1.3. We denote these algebras by $\mathcal{T}(\Sigma, E)$ and $\mathcal{T}_0(\Sigma, E)$, and call them the *term model* and *ground term model*, respectively.

The term model is the key to the completeness result below. The ground term model is of interest because of its initiality property, i.e. the elements are the closed terms, and terms denote equal elements only if the equations force them to.

7.1.12. Exercise. Prove the following two substitution lemmas.

(i) Let $\mathcal{A}$ be a Σ-algebra, $\upsilon : Var \to A$ be an assignment in $\mathcal{A}$ and let σ be a substitution. Then for all $s, t \in Ter(\Sigma)$,

$$[\![t^\sigma]\!]^{\mathcal{A},\upsilon} = [\![t]\!]^{\mathcal{A},[\![\sigma]\!]^{\mathcal{A},\upsilon}}$$

where $[\![\sigma]\!]^{\mathcal{A},\upsilon}$ is the assignment which assigns $[\![\sigma(x)]\!]^{\mathcal{A},\upsilon}$ to every variable x.

(ii) Let (Σ, E) be an equational specification, and let the assignment $\upsilon : Var \to Ter(\Sigma)/=_E$ and the substitution $\sigma : Var \to Ter(\Sigma)$ be such that $[\sigma(x)]_{=_E} = \upsilon(x)$ for every variable x. Then $[\![t]\!]^{\mathcal{T}(\Sigma,E),\upsilon} = [t^\sigma]_{=_E}$ for every term $t \in Ter(\Sigma)$.

7.1.13. Proposition. *For every equational specification (Σ, E), $\mathcal{T}(\Sigma, E) \models E$ and $\mathcal{T}_0(\Sigma, E) \models E$.*

Proof. We show that the term model is a model of E. The reasoning for the ground term model is entirely analogous. We have to show that for every $s = t \in E$ and every assignment υ in $\mathcal{T}(\Sigma, E)$, $[\![s]\!]^{\mathcal{T}(\Sigma,E),\upsilon} = [\![t]\!]^{\mathcal{T}(\Sigma,E),\upsilon}$.

Thus let $s = t \in E$ and υ be arbitrary, and pick a substitution σ such that $\sigma(x) \in \upsilon(x)$ for every variable x, i.e. $[\sigma(x)]_{=_E} = \upsilon(x)$ for every variable x. By the second substitution lemma in the preceding exercise we then have that $[\![t']\!]^{\mathcal{T}(\Sigma,E),\upsilon} = [t'^{\sigma}]_{=_E}$ for every $t' \in \mathit{Ter}(\Sigma)$. Now, since $s = t \in E$, we have $s^{\sigma} =_E t^{\sigma}$ by (sub). Hence

$$[\![s]\!]^{\mathcal{T}(\Sigma,E),\upsilon} = [s^{\sigma}]_{=_E} = [t^{\sigma}]_{=_E} = [\![t]\!]^{\mathcal{T}(\Sigma,E),\upsilon} \qquad \square$$

Given an equational specification, the *soundness* of equational logic asserts that applying its axioms and rules for deducing new equations always yields equations that are valid in every model of the equational specification. Conversely, *completeness* asserts that every equation valid in every model of a given equational specification can be deduced using the axioms and rules. These two properties together imply that, for the class of models of a given equational specification, the model theoretic notion of an equation being valid in all algebras in the class coincides with the proof theoretic notion of the equation being derivable from the given equations by means of equational logic. This theorem, which was first given by Birkhoff [1935], is proved below by combining the following two propositions.

7.1.14. Notation. Let (Σ, E) be an equational specification and let $s, t \in \mathit{Ter}(\Sigma)$. If $s = t$ is valid in every model of E, we write $(\Sigma, E) \models s = t$.

7.1.15. Proposition (soundness). *Let (Σ, E) be an equational specification. Then for all $s, t \in \mathit{Ter}(\Sigma)$, if $(\Sigma, E) \vdash s = t$ then $(\Sigma, E) \models s = t$.*

Proof. Let (Σ, E) be an equational specification and suppose $(\Sigma, E) \vdash s = t$. We shall employ induction on the length of a derivation. If $s = t$ is derivable by (E) or (ref), the result is immediate. For the induction step we distinguish four cases according to the last rule applied. For (sym), (trans) and (F) this is routine. Thus suppose $s \equiv s'^{\sigma}$, $t \equiv t'^{\sigma}$ and $(\Sigma, E) \vdash s = t$ is obtained by (sub). To prove $(\Sigma, E) \models s = t$, we have to show for an arbitrary Σ-algebra $\mathcal{A}$ and an arbitrary assignment υ in $\mathcal{A}$ that $[\![s]\!]^{\mathcal{A},\upsilon} = [\![t]\!]^{\mathcal{A},\upsilon}$. Thus let $\mathcal{A}$ and υ be arbitrary. Then

$$\begin{array}{lcll} [\![s]\!]^{\mathcal{A},\upsilon} = [\![s'^{\sigma}]\!]^{\mathcal{A},\upsilon} & = & [\![s']\!]^{\mathcal{A},[\![\sigma]\!]^{\mathcal{A},\upsilon}} & \text{by Exercise 7.1.12(i)} \\ & = & [\![t']\!]^{\mathcal{A},[\![\sigma]\!]^{\mathcal{A},\upsilon}} & \text{since } (\Sigma, E) \models s' = t' \\ & = & [\![t'^{\sigma}]\!]^{\mathcal{A},\upsilon} & \text{again by Exercise 7.1.12(i)} \\ & = & [\![t]\!]^{\mathcal{A},\upsilon} & \end{array} \qquad \square$$

7.1.16. Proposition (completeness). *Let (Σ, E) be an equational specification. Then for all $s, t \in \mathit{Ter}(\Sigma)$, if $(\Sigma, E) \models s = t$ then $(\Sigma, E) \vdash s = t$.*

Proof. Suppose $(\Sigma, E) \models s = t$. Then $\mathcal{T}(\Sigma, E) \models s = t$. Thus $[\![s]\!]^{\mathcal{T}(\Sigma,E),\upsilon} = [\![t]\!]^{\mathcal{T}(\Sigma,E),\upsilon}$ where $\upsilon(x) = [x]_{=_E}$ for every variable x. Note that for this assignment we have by Exercise 7.1.12(ii) that $[\![t']\!]^{\mathcal{T}(\Sigma,E),\upsilon} = [t']_{=_E}$ for every $t' \in \mathit{Ter}(\Sigma)$. Hence $[s]_{=_E} = [t]_{=_E}$ and therefore $E \vdash s = t$. $\square$

Combining both propositions yields Birkhoff's Theorem.

7.1.17. THEOREM (Birkhoff [1935]). *Let (Σ, E) be an equational specification. Then for all $s, t \in Ter(\Sigma)$, $(\Sigma, E) \vdash s = t$ if and only if $(\Sigma, E) \models s = t$.*

7.1.18. EXERCISE. Let (Σ, E) be the specification of the natural numbers with addition given in Example 7.1.2. Show that $(\Sigma, E) \nvdash Add(x, y) = Add(y, x)$. (Hint: Consider any non-commutative group $\mathcal{G} = \langle G, +, 0\rangle$. Make $\mathcal{G}$ into a Σ-algebra by defining $S^{\mathcal{G}}(g) = g$ for every $g \in G$. Now show that $\mathcal{G}$ is a model of E and apply Birkhoff's Theorem.)

As already mentioned in Section 2.4, the *validity problem* or *uniform word problem* for (Σ, E) is the problem:

Given $s, t \in Ter(\Sigma)$, decide whether or not $(\Sigma, E) \models s = t$.

According to Birkhoff's completeness theorem for equational logic this amounts to deciding whether $(\Sigma, E) \vdash s = t$. Complete TRSs (i.e. TRSs which are SN and CR) may solve this problem.

7.1.19. DEFINITION. Let (Σ, E) be an equational specification and let (Σ, R) be a TRS. (Σ, R) is a *TRS for* (Σ, E) if $=_R$ and $=_E$ coincide, i.e. if for all terms $s, t \in Ter(\Sigma)$,

$$s =_R t \Leftrightarrow s =_E t$$

Now suppose we can find a complete TRS (Σ, R) for the equational specification (Σ, E). Then, provided R has only finitely many rewrite rules, we have a positive solution of the validity problem. The decision algorithm is simple:

(i) *reduce s and t to their respective normal forms s', t';*
(ii) *compare s' and t': $s =_R t$ iff $s' \equiv t'$.*

We now have to face the question of how to find a complete TRS (Σ, R) for a given specification (Σ, E). This is not possible in general, since not every E (even if finite) has a decidable validity problem. An example of such an E with undecidable validity problem is the set of equations in Table 7.2 which is obtained from CL, combinatory logic, introduced in Section 3.2, by replacing '$\rightarrow$' by '$=$'. For a proof of the unsolvability see Barendregt [1984].

$Sxyz$	$=$	$xz(yz)$
Kxy	$=$	x
Ix	$=$	x

Table 7.2: Equational specification of CL

So the validity problem of (Σ, E) can be solved if a complete TRS (Σ, R) for (Σ, E) can be provided. The reverse statement does not hold: there are equational specifications with decidable validity problem but without a complete TRS (see Exercise 2.6.2). Restricting attention to ground equations, we are

considering the *word problem* for (Σ, E), which is the following decidability problem:

Given $s, t \in Ter_0(\Sigma)$, decide whether or not $(\Sigma, E) \models s = t$.

Similarly to the case of the uniform word problem, complete TRSs provide a positive solution for the word problem. However, since we require less than completeness, namely SN and CR only for ground terms, it may be that a complete TRS for (Σ, E) cannot be found, while there does exist a TRS which solves the problem (see e.g. Exercise 2.6.2). Note that there are finite equational specifications with decidable validity problem for which not even a TRS exists which is complete with respect to ground terms. This strengthens the observation in Exercise 2.6.2. The simplest such specification consists of a constant and a single binary commutative operator. According to Exercise 2.6.2, which also works for the present simpler specification, no complete TRS (Σ, R) can be found such that for all open terms s, t we have $s =_R t \Leftrightarrow s =_E t$. According to the next exercise, we also have the stronger result that no TRS (Σ, R) exists which is ground complete and such that $=_R$ and $=_E$ coincide on $Ter_0(\Sigma)$.

7.1.20. Exercise (Bergstra and Klop). Let (Σ, E) be the specification with $\Sigma = \{0, Times\}$ and $E = \{Times(x, y) = Times(y, x)\}$. The decidability of the validity problem for (Σ, E) follows in a way quite similar to Exercise 2.6.2. Prove that there is no finite TRS (Σ, R) such that the restriction to ground terms, $(\Sigma, R)_0$, is complete and such that $=_R$ and $=_E$ coincide on ground terms. (Hint: Define terms $t_0 \equiv 0$, $t_{n+1} \equiv Times(t_n, t_n)$ $(n \geq 0)$. Suppose (Σ, R) is a TRS with finitely many rewrite rules such that $=_R$ and $=_E$ coincide on ground terms. Let N be the maximal depth of the term trees of left-hand sides of rewrite rules in R. Now consider the terms $t^* \equiv Times(t_N, t_{2N})$ and $t^{**} \equiv Times(t_{2N}, t_N)$.)

7.2. Initial algebra semantics

Initial algebras provide standard models of equational specifications. These contain only elements that can be constructed from those appearing in the specification, and satisfy only those closed equations that appear in the specification or are logical consequences of them.

7.2.1. Definition. Let (Σ, E) be an equational specification.

(i) Let $\mathcal{A} = \langle A, \{F^{\mathcal{A}} \mid F \in \Sigma\}\rangle$, $\mathcal{B} = \langle B, \{F^{\mathcal{B}} \mid F \in \Sigma\}\rangle$ be Σ-algebras. $h : A \to B$ is a Σ-*homomorphism* from $\mathcal{A}$ to $\mathcal{B}$ if for each n-ary $F \in \Sigma$ and all $a_1, \ldots, a_n \in A$,

$$h(F^{\mathcal{A}}(a_1, \ldots, a_n)) = F^{\mathcal{B}}(h(a_1), \ldots, h(a_n))$$

If h is bijective then we write $\mathcal{A} \cong \mathcal{B}$, say that $\mathcal{A}$ and $\mathcal{B}$ are *isomorphic* and call h a Σ-*isomorphism*.

(ii) Let $\mathcal{A}$ be a model of E. $\mathcal{A}$ is said to be *initial in* (the class of all models of) E if for every model $\mathcal{B}$ of E, there exists a unique Σ-homomorphism from $\mathcal{A}$ to $\mathcal{B}$.

7.2.2. EXAMPLE. Consider, as in Example 7.1.2, the simple specification (Σ, E) of the natural numbers with addition where $\Sigma = \{0, S, Add\}$ and E consists of the two equations

$$\begin{aligned} Add(x, 0) &= x \\ Add(x, S(y)) &= S(Add(x, y)) \end{aligned}$$

Let $\mathcal{N}$ be the Σ-algebra of natural numbers with $0^{\mathcal{N}} = 0$, $S^{\mathcal{N}} = \lambda n \in \mathbb{N}.n + 1$ and $Add^{\mathcal{N}} = \lambda n, m \in \mathbb{N}.n + m$. Then $\mathcal{N}$ is initial in E: $\mathcal{N}$ is clearly a model of E. Let $\mathcal{A}$ be a model of E and define $h : \mathbb{N} \to A$ inductively by $h(0) = 0^{\mathcal{A}}$ and $h(n+1) = S^{\mathcal{A}}(h(n))$. One can easily show that h is a homomorphism from $\mathcal{N}$ to $\mathcal{A}$. Now let h' be any homomorphism from $\mathcal{N}$ to $\mathcal{A}$. We prove by induction on $n \in \mathbb{N}$ that h and h' coincide. Since h' is a homomorphism, we have $h'(0) = h'(0^{\mathcal{N}}) = 0^{\mathcal{A}} = h(0)$. Suppose $h'(n) = h(n)$. Then we have, applying again the fact that h' is a homomorphism, that $h'(n+1) = h'(S^{\mathcal{N}}(n)) = S^{\mathcal{A}}(h'(n)) = S^{\mathcal{A}}(h(n)) = h(n+1)$. This shows that h is unique.

Perhaps the most basic fact about initial algebras is that any two are abstractly the same, in that they differ only in the representations given to their elements.

7.2.3. EXERCISE. Let (Σ, E) be an equational specification, and let $\mathcal{A} = \langle A, \{F^{\mathcal{A}} \mid F \in \Sigma\}\rangle$, $\mathcal{B} = \langle B, \{F^{\mathcal{B}} \mid F \in \Sigma\}\rangle$ be Σ-algebras.

(i) Show that $\mathcal{A} \cong \mathcal{B}$ if and only if there exist Σ-homomorphisms $h : A \to B$, $h' : B \to A$ such that $h'(h(a)) = a$ and $h(h'(b)) = b$ for all $a \in A$, $b \in B$.

(ii) Show that initial models in E are unique up to isomorphism, i.e. show that if $\mathcal{A}$ and $\mathcal{B}$ are initial in E, then $\mathcal{A} \cong \mathcal{B}$.

7.2.4. NOTATION. Although the initial algebra is only determined up to isomorphism, we will speak of 'the' initial algebra and use the notation $I(\Sigma, E)$ for it.

7.2.5. EXERCISE. Let (Σ, E) be an equational specification. Quite similarly to the proof given in Example 7.2.2 one can show that $I(\Sigma, E) \cong \mathcal{T}_0(\Sigma, E)$. Draw up a detailed proof.

7.2.6. EXERCISE. An equational specification (Σ, E) is said to be *ω-complete* iff for all $s, t \in Ter(\Sigma)$, $(\Sigma, E) \models s = t \Leftrightarrow I(\Sigma, E) \models s = t$. Clearly every equation that is provable from the axioms in E is also valid in the initial algebra. The converse of this statement, however, does not need to hold. Give an example of such an ω-incomplete specification. (Hint: Consider the specification of groups as given in Table 7.3 below. Show that $(\Sigma, E) \not\models x \cdot y = y \cdot x$. Commutativity of $\cdot$, however, is valid in the initial algebra.)

7.3. Critical pair completion

We resume the question of how to find – if it exists – a complete TRS (for the case of open terms, henceforth) for an equational specification (Σ, E). This is in fact what the Knuth–Bendix completion algorithm is trying to do. We will now explain the essential features of the completion algorithm first by an informal, "intuition-guided" completion of the equational specification of groups in Table 7.3.

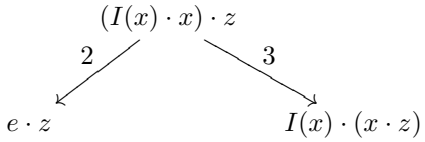
$e \cdot x$	$=$	x
$I(x) \cdot x$	$=$	e
$(x \cdot y) \cdot z$	$=$	$x \cdot (y \cdot z)$

Table 7.3: Equational specification of groups

First we give these equations a 'sensible' orientation:

(1) $e \cdot x \to x$,
(2) $I(x) \cdot x \to e$,
(3) $(x \cdot y) \cdot z \to x \cdot (y \cdot z)$.

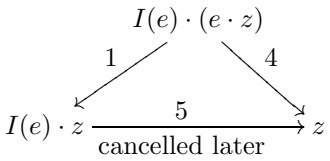
Note that the orientation in rules (1), (2) is forced, by the restrictions on rewrite rules in Section 2.2. As to the orientation of rule (3), the other direction is just as 'sensible'. These rules are not confluent, as can be seen by superposition of e.g. (2) and (3). Redex $I(x) \cdot x$ can be unified with the underlined subterm of redex $\underline{(x \cdot y)} \cdot z$. The term $(I(x) \cdot x) \cdot z$ is consequently subject to two possible reductions:



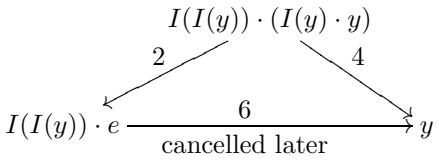
The pair of reducts $\langle e \cdot z, I(x) \cdot (x \cdot z) \rangle$ is a non-convergent critical pair: $I(x) \cdot (x \cdot z)$ is a normal form and $e \cdot z$ only reduces to the normal form z. So we have the problematic pair of terms $\langle z, I(x) \cdot (x \cdot z) \rangle$; problematic because their equality is derivable from E, but they have no common reduct with respect to the reduction available so far. Therefore we adopt a new rule

(4) $I(x) \cdot (x \cdot z) \to z$.

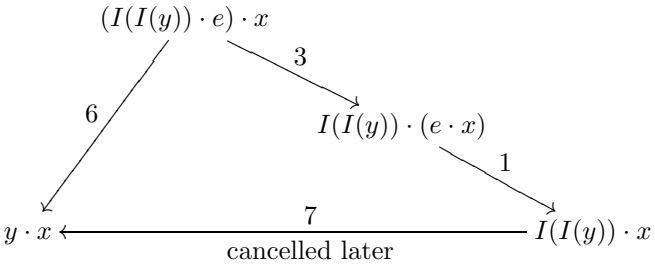
Now we have overlap of the rules (4) and (1):



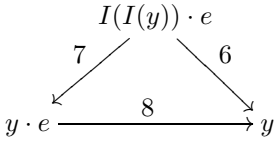
This yields the critical pair $\langle I(e) \cdot z, z \rangle$ which cannot further be reduced. We now adopt step 5 (see picture) as a new rule (5) which, as will turn out at a later stage, will become superfluous. We go on searching for critical pairs and find the overlap of (4) and (2):



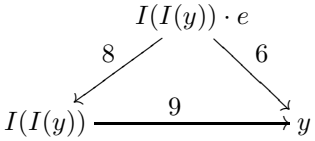
Overlap of (3) and (6):



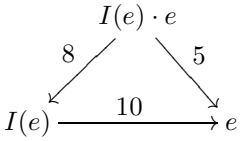
Overlap of (6) and (7):



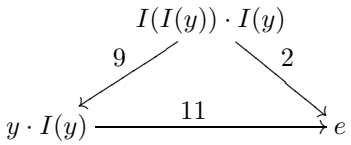
Overlap of (6) and (8):



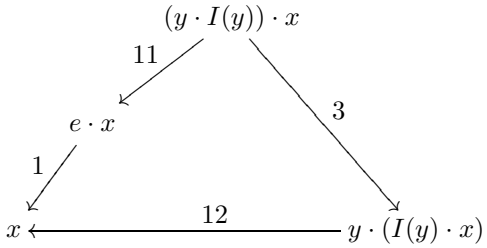
Rule (6) is now no longer necessary to ensure that the critical pair $\langle I(I(y)) \cdot e, y \rangle$ has a common reduct, because $I(I(y)) \cdot e \to_9 y \cdot e \to_8 y$. Likewise for rule (7). Overlap of (8) and (5):

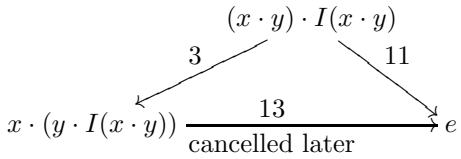


Rule (5) can now be cancelled. Overlap of (2) and (9):

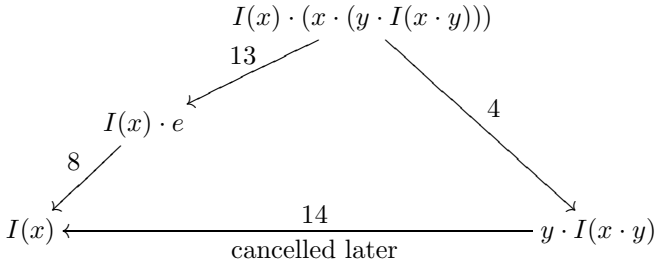


Two overlaps between (3) and (11):

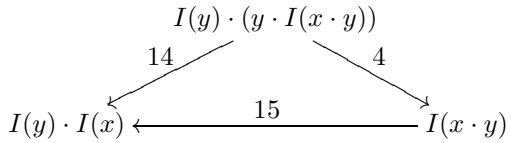




Overlap of (4) and (13):



Rule (13) can now be cancelled. Overlap of (4) and (14):



Now rule (14) can be cancelled. At this moment, the TRS with rewrite rules as in Table 7.4 has only convergent critical pairs.

Furthermore, one can prove SN for this TRS by a Knuth–Bendix ordering or by the recursive path ordering. In fact we need the extended lexicographic version, due to the presence of the associativity rule. According to Theorem 2.7.16 the TRS is therefore CR and hence complete. We conclude that the validity problem for the equational specification of groups is decidable.

(1)	$e \cdot x$	$\to$	x	(9)	$I(I(y))$	$\to$	y
(2)	$I(x) \cdot x$	$\to$	e	(10)	$I(e)$	$\to$	e
(3)	$(x \cdot y) \cdot z$	$\to$	$x \cdot (y \cdot z)$	(11)	$y \cdot I(y)$	$\to$	e
(4)	$I(x) \cdot (x \cdot z)$	$\to$	z	(12)	$y \cdot (I(y) \cdot x)$	$\to$	x
(8)	$y \cdot e$	$\to$	y	(15)	$I(x \cdot y)$	$\to$	$I(y) \cdot I(x)$

Table 7.4: A complete TRS for the equational specification of groups

7.3.1. Exercise (Knuth and Bendix [1970]). Table 7.5 contains the specifications (Σ, E_{L-R}) and (Σ, E_{R-L}), called *L-R theory* and *R-L theory*, which are variations of the specification (Σ, E_G) of groups in Table 7.4. Prove, using the completions in

L-R theory

$e \cdot x$	$=$	x
$x \cdot I(x)$	$=$	e
$(x \cdot y) \cdot z$	$=$	$x \cdot (y \cdot z)$

R-L theory

$x \cdot e$	$=$	x
$I(x) \cdot x$	$=$	e
$(x \cdot y) \cdot z$	$=$	$x \cdot (y \cdot z)$

Completions

$e \cdot x$	$\to$	x	$x \cdot e$	$\to$	x
$x \cdot I(x)$	$\to$	e	$I(x) \cdot x$	$\to$	e
$(x \cdot y) \cdot z$	$\to$	$x \cdot (y \cdot z)$	$(x \cdot y) \cdot z$	$\to$	$x \cdot (y \cdot z)$
$I(e)$	$\to$	e	$I(e)$	$\to$	e
$I(x \cdot y)$	$\to$	$I(y) \cdot I(x)$	$I(x \cdot y)$	$\to$	$I(y) \cdot I(x)$
$x \cdot (I(x) \cdot y)$	$\to$	y	$x \cdot (y \cdot I(y))$	$\to$	x
$I(x) \cdot (x \cdot y)$	$\to$	y	$I(x) \cdot (x \cdot y)$	$\to$	$I(I(y))$
$x \cdot e$	$\to$	$I(I(x))$	$e \cdot x$	$\to$	$I(I(x))$
$I(I(I(x)))$	$\to$	$I(x)$	$I(I(I(x)))$	$\to$	$I(x)$
$I(I(x)) \cdot y$	$\to$	$x \cdot y$	$x \cdot I(I(y))$	$\to$	$x \cdot y$
			$x \cdot (I(I(y)) \cdot z)$	$\to$	$x \cdot (y \cdot z)$
			$x \cdot (y \cdot (I(y) \cdot z))$	$\to$	$x \cdot z$

Table 7.5: Specifications and completions of L-R and R-L theory

Table 7.5, that $x \cdot e = x$ is not derivable in L-R theory and that in R-L theory the equations $e \cdot x = x$ and $x \cdot I(x) = e$ are not derivable. Hence the three theories are different, i.e. determine different classes of models. In fact, note that

(i) $\mathcal{A} \models E_G \Leftrightarrow \mathcal{A} \models E_{R-L} \cup E_{L-R}$,
(ii) $\exists \mathcal{A}\ \mathcal{A} \models E_{R-L} \ \wedge \ \mathcal{A} \not\models E_{L-R}$,
(iii) $\exists \mathcal{A}\ \mathcal{A} \models E_{L-R} \ \wedge \ \mathcal{A} \not\models E_{R-L}$.

The completion procedure above by hand was naive, since we were not very systematic in searching for critical pairs, and especially since we were guided by an intuitive sense only of what direction to adopt when generating a new rule. In most cases there was no other possibility (e.g. at (4): $z \to I(x) \cdot (x \cdot z)$ is not a reduction rule due to the restriction that the left-hand side is not a single variable), but in case (15) the other direction was at least as plausible, as it is even length-decreasing. However, the other direction $I(y) \cdot I(x) \to I(x \cdot y)$ would have led to disastrous complications (described in Knuth and Bendix [1970]). The problem of what direction to choose is solved in the actual Knuth–Bendix algorithm and its variants by preordaining a reduction ordering on the terms, as defined in Definition 6.1.2. Recall that we have by Proposition 6.1.3 that a TRS (Σ, R) is terminating if and only if it admits a compatible reduction order, i.e. a reduction order $<$ on $Ter(\Sigma)$ such that $l > r$ for every rewrite rule $l \to r$ in R. However, finding a reduction ordering that can serve as an input to the algorithm is a matter of ingenuity or experimentation.

Input: – an equational specification (Σ, E)
– a reduction ordering $<$ on Ter(Σ)
(i.e. a program which computes $<$)
Output: – a complete TRS R such that for all $s, t \in Ter(\Sigma)$,
$s =_R t \Leftrightarrow (\Sigma, E) \vdash s = t$

$E_0 := E$; $R_0 := \varnothing$; $i := 0$
while $E_i \neq \varnothing$ **do**
  choose an equation $s = t \in E_i$;
  reduce s and t to respective R_i-normal forms s' and t';
  if $s' \equiv t'$ **then** $E_{i+1} := E_i - \{s = t\}$; $R_{i+1} := R_i$; $i := i + 1$
  else
    if $s' > t'$ **then** $l := s'$; $r := t'$
    else if $t' > s'$ **then** $l := t'$; $r := s'$
    else *failure*
    fi;
    $R_{i+1} := R_i \cup \{l \to r\}$;
    $CP := \{s'' = t'' \mid \langle s'', t'' \rangle$ is a critical pair between
        the rules in R_{i+1} and $l \to r\}$;
    $E_{i+1} := E_i \cup CP - \{s = t\}$;
    $i := i + 1$
  fi;
od;
success

Table 7.6: Knuth–Bendix completion algorithm

In Table 7.6 a simple version of the Knuth–Bendix completion algorithm is presented. The program has three possibilities: it may

(i) terminate successfully,

(ii) loop indefinitely, or

(iii) fail because a pair of terms s, t cannot be oriented (i.e. neither $s > t$ nor $t > s$).

The third case gives the most important restriction of the Knuth–Bendix algorithm: equational specifications with commutative operators cannot be completed. If one still wants to deal with equational specifications having commutative or associative operators as in e.g. Exercise 6.2.11, one has to work modulo the equations of associativity and commutativity. For completion modulo such equations we refer to Peterson and Stickel [1981] and Jouannaud and Kirchner [1986].

In the first case, the algorithm behaves correctly, in that it produces a complete TRS for the input specification.

7.3.2. THEOREM. *If the Knuth–Bendix completion algorithm terminates with success on input (Σ, E), then the output consists of a complete TRS for (Σ, E).*

PROOF. Suppose the algorithm stops successfully at iteration i on input (Σ, E) computing the successive specifications $E_0, \ldots, E_i$ and TRSs $R_0, \ldots, R_i$. Denoting the convertibility $=_{E_n} \cup =_{R_n}$ by $=_n$, we shall show that for all $0 \leq n \leq i$,

(i) $=_n = =_E$,

(ii) for all critical pairs $\langle \tilde{s}, \tilde{t} \rangle$ of R_n, there exist $\hat{s}, \hat{t} \in \mathit{Ter}(\Sigma)$ such that $\tilde{s} \twoheadrightarrow_{R_n} \hat{s}$, $\tilde{t} \twoheadrightarrow_{R_n} \hat{t}$, and $\hat{s} \equiv \hat{t}$ or $\hat{s} = \hat{t} \in E_n$.

Once (i) and (ii) are proved, the theorem follows immediately: For, since $E_i = \varnothing$, it follows from (i) that $=_{R_i} = =_E$, and from (ii) that all critical pairs are convergent. Thus (Σ, R_i) is a weakly confluent TRS for (Σ, E) by Lemma 2.7.15. SN follows from the fact that $<$ is compatible with (Σ, R_i). Hence (Σ, R_i) is also CR by Newman's Lemma, Theorem 1.2.1, and whence complete. We shall now prove (i) and (ii) by induction. The base case is trivial. Suppose that (i), (ii) hold for n and that at iteration $n + 1$ the equation $s = t \in E_n$ is chosen with respective R_n-normal forms s', t'.

Case $s' \equiv t'$: Then $E_{n+1} = E_n - \{s = t\}$ and $R_{n+1} = R_n$. To prove (i), it suffices to show that $=_{n+1} = =_n$. We clearly have $=_{n+1} \subseteq =_n$. The inverse inclusion follows from the fact that $s =_{R_{n+1}} t$. For (ii) let $\langle \tilde{s}, \tilde{t} \rangle$ be a critical pair of R_{n+1}. Since $R_{n+1} = R_n$, it follows from the induction hypothesis that there exist $\hat{s}, \hat{t} \in \mathit{Ter}(\Sigma)$ such that $\tilde{s} \twoheadrightarrow_{R_{n+1}} \hat{s}$, $\tilde{t} \twoheadrightarrow_{R_{n+1}} \hat{t}$, and $\hat{s} \equiv \hat{t}$ or $\hat{s} = \hat{t} \in E_n$. If $\hat{s} \equiv \hat{t}$ or $\hat{s} = \hat{t} \in E_{n+1}$, we are done. Thus assume $\hat{s} = \hat{t} \in E_n - E_{n+1}$. Then $\hat{s} \equiv s$ and $\hat{t} \equiv t$. But then $\tilde{s} \twoheadrightarrow_{R_{n+1}} s'$, $\tilde{t} \twoheadrightarrow_{R_{n+1}} t'$ and $s' \equiv t'$.

Case $s' \not\equiv t'$: We assume that $s' > t'$ (the proof for $t' > s'$ proceeds similarly). Then $R_{n+1} = R_n \cup \{s' \to t'\}$ and $E_{n+1} = E_n \cup CP - \{s = t\}$ where CP consists of the critical pairs between the rules in R_{n+1} and $s' \to t'$. For (i), observe that $s =_{R_{n+1}} t$ and therefore $=_n \subseteq =_{n+1}$. Moreover, since every pair in CP is R_n-convertible and $s = t \in E_n$, we also have $=_{n+1} \subseteq =_n$. To prove (ii), let $\langle \tilde{s}, \tilde{t} \rangle$ be a critical pair of R_{n+1}. If $\langle \tilde{s}, \tilde{t} \rangle$ is also critical in R_n, it follows from the induction hypothesis that $\tilde{s}$ and $\tilde{t}$ have R_n-reducts $\hat{s}$ and $\hat{t}$ with $\hat{s} \equiv \hat{t}$ or $\hat{s} = \hat{t} \in E_n$. In case $\hat{s} \equiv \hat{t}$ or $\hat{s} = \hat{t} \in E_{n+1}$, we are done, since R_n-reducts are also R_{n+1}-reducts. Otherwise, $\hat{s} \equiv s$ and $\hat{t} \equiv t$ and therefore $\tilde{s} \twoheadrightarrow_{R_{n+1}} t' \twoheadleftarrow_{R_{n+1}} \tilde{t}$. Finally, if $\langle \tilde{s}, \tilde{t} \rangle$ is not critical in R_n, then it is a critical pair between the rules in R_{n+1} and $s' \to t'$. Whence $\tilde{s} = \tilde{t} \in E_{n+1}$. □

7.3.3. Exercise. Let (Σ, E_1) and (Σ, E_2) be the equational specifications with $E_1 = \{F(F(x)) = G(x)\}$ and $E_2 = \{F(G(F(x))) = G(F(x))\}$.

(i) Apply the Knuth–Bendix completion algorithm to (Σ, E_1) and compute three different complete TRSs for it.

(ii) Show that the completion algorithm, when applied to (Σ, E_2), does not terminate.

A

Mathematical background

Marc Bezem

This appendix provides a self-contained introduction to basic mathematical concepts and results used throughout the book. We treat successively: relations, functions and orders; multisets.

In this appendix, we provide the necessary background in mathematics. We will assume that the reader is familiar with the terminology and notation of elementary set theory and logic. Good sources are Enderton [1977] and van Dalen [1994], respectively. We use $\varnothing$, $\in$, $\cup$, $\cap$, $-$, $\times$, $\subset$, $\subseteq$, $\to$, $\mathcal{P}$ for the *empty set, membership, union, intersection, difference, cartesian product, (strict) inclusion, function space, power set*, respectively. The sets of natural numbers $\{0, 1, 2, \ldots\}$ and integers $\{\ldots, -1, 0, 1, \ldots\}$ are denoted by $\mathbb{N}$ and $\mathbb{Z}$, respectively. Given a set S, the elements of S having property ϕ form a subset of S denoted by $\{s \in S \mid \phi(s)\}$. Furthermore, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ denote logical *negation, and, or, implication, equivalence*, respectively. We use $\forall$ ($\exists$) for *universal (existential)* quantification. The quantifiers and $\neg$ bind strongest. The operators $\wedge, \vee$ bind more strongly than $\Rightarrow, \Leftrightarrow$. Thus, for example, $\neg\forall x\ \phi(x) \Rightarrow \psi \vee \chi$ stands for $(\neg(\forall x\ \phi(x))) \Rightarrow (\psi \vee \chi)$.

A.1. Relations, functions and orders

We start by defining a number of important properties of relations. If $R \subseteq A \times B$ for sets A and B, then R is called a *(binary) relation* and we also write $R(a, b)$ and $a\,R\,b$ for $(a, b) \in R$. If, moreover, $R' \subseteq B \times C$ is a binary relation, then $a\ R\ b\ R'\ c$ is shorthand for $(a, b) \in R \wedge (b, c) \in R'$. If $A = B$, then R is called a *(binary) relation on* A. Functions and orders will be introduced as special cases of relations.

A.1.1. Definition. Let $R \subseteq A \times B$ for sets A and B.

(i) R is *single-valued* if $\forall a \in A\ \forall b, b' \in B\ (R(a, b) \wedge R(a, b') \Rightarrow b = b')$. In that case R is also called a *partial function* from A to B, and we write $R(a)$ for the b such that $R(a, b)$, provided such b exists.

(ii) R is a *total function* if R is single-valued and $\forall a \in A\ \exists b \in B\ R(a, b)$. A total function is also simply called a *function*, *mapping* or *map*, and the set of all functions from A to B is denoted by $A \to B$. We write $f : A \to B$

to express that f is a function from A to B. Furthermore, $f(X)$ denotes the subset $\{f(x) \mid x \in X\}$ of B, for every $X \subseteq A$. Also, $f^{-1}(Y)$ denotes the subset $\{y \in A \mid f(y) \in Y\}$ of A, for every $Y \subseteq B$. The sets $f(X)$ and $f^{-1}(Y)$ are called the *image* of X under f and the *inverse image* of Y under f, respectively.

(iii) For sets A, B, C and relations $R \subseteq A \times B$ and $R' \subseteq B \times C$, the *composition* of R and R' is the relation $R' \circ R \subseteq A \times C$ defined by $R' \circ R = \{(a, c) \mid \exists b \in B\ (R(a, b) \wedge R'(b, c))\}$. Note that $R' \circ R$ is single-valued if both R and R' are. In particular, $R' \circ R$ is a total function if both R and R' are. In that case we have $(R' \circ R)(a) = R'(R(a))$ for all $a \in A$. Mind the order in the notation $R' \circ R$. Sometimes this order is inconvenient. Therefore we also denote the composition of R and R' by $R \cdot R'$, where the order corresponds to the order in which the relations are applied. Thus we have $R \cdot R' = R' \circ R$.

(iv) If $R \subseteq A \times A$, then the n-fold composition of R with itself is denoted by R^n. More precisely, $R^0 = \{(a, a) \mid a \in A\}$ and $R^{n+1} = R \circ (R^n) = (R^n) \circ R = R \circ \cdots \circ R$ ($n + 1$ times R). If R is single-valued, then R^n is so for all $n \in \mathbb{N}$. In particular, if R is a function, then R^n is a function called the nth *iteration* of R.

(v) The *inverse* of R is the relation $R^{-1} \subseteq B \times A$ defined by $R^{-1}(b, a) \Leftrightarrow R(a, b)$ for all $a \in A,\ b \in B$.

A.1.2. Definition. Let f be a function from A to B.

(i) $f : A \to B$ is *injective* if $\forall a, a' \in A\ (f(a) = f(a') \Rightarrow a = a')$. An injective function is also called an *injection*.

(ii) $f : A \to B$ is *surjective* if $\forall b \in B\ \exists a \in A\ (f(a) = b)$. A surjective function is also called a *surjection*.

(iii) $f : A \to B$ is *bijective* if f is both injective and surjective. A bijective function is also called a *bijection*.

(iv) If $f : A \to B$ is bijective, then the inverse relation f^{-1} is also a function. In that case we call $f^{-1} : B \to A$ the *inverse function* of f, and f^{-1} is also a bijection and satisfies $\forall a \in A\ f^{-1}(f(a)) = a$ and $\forall b \in B\ f(f^{-1}(b)) = b$.

(v) The *identity function* on A is the function $id_A : A \to A$ defined by $id_A(a) = a$ for all $a \in A$.

(vi) If $f : A \to A$, then a *fixed point* of f is an element $a \in A$ such that $f(a) = a$.

A.1.3. Exercise. Justify the claims in Definitions A.1.1(iii) and A.1.2(iv). Prove $(R_1 \circ R_2)^{-1} = R_2^{-1} \circ R_1^{-1}$ and $R_1 \circ (R_2 \circ R_3) = (R_1 \circ R_2) \circ R_3$.

A.1.4. Definition. Let S be a set and $R \subseteq S \times S$ a relation on S.

(i) A *sequence* in S is a function $s : \mathbb{N} \to S$. A sequence s is also denoted by $s_0, s_1, s_2, \ldots$, where s_i denotes $s(i)$.

(ii) A *finite sequence* of *length* n in S is a function $s : \{0, 1, \ldots, n-1\} \to S$ ($n \geq 0$), also denoted by $\langle s_0, s_1, \ldots, s_{n-1}\rangle$, or $s_0 s_1 \ldots s_{n-1}$. The set of finite sequences of length n is denoted by S^n. The *empty sequence* is denoted by $\langle\rangle$. Finite sequences of arbitrary length are also called *lists*. The set of all finite sequences in S is denoted by S^*. *Concatenation* of finite sequences is denoted by $\cdot$.

(iii) A sequence s in S is *R-descending*, or simply *descending* when the relation R is clear from the context, if $R(s(i+1), s(i))$ for all $i \in \mathbb{N}$. Similarly, s is *R-ascending* if $R(s(i), s(i+1))$ for all $i \in \mathbb{N}$.

(iv) A *subsequence* of a sequence s is the composition $s \circ i$ with an ascending sequence i in $\mathbb{N}$, that is, $s_{i_0}, s_{i_1}, s_{i_2}, \ldots$ with $i_0 < i_1 < i_2 < \cdots$. A *finite* subsequence of s is a finite sequence of the form $\langle s_{i_1}, \ldots, s_{i_k}\rangle$, where $k \geq 0$ and $i_1 < \cdots < i_k$.

The subscript of an element of a (finite) sequence is called the *index* of that element. The notions ascending, descending and subsequence also apply to finite sequences, taking into account the finite range of the index.

A.1.5. Definition. Let S be a set and $R \subseteq S \times S$ a relation on S.

(i) R is *reflexive* if $\forall x \in S\ R(x, x)$.

(ii) R is *irreflexive* if $\forall x \in S\ \neg R(x, x)$.

(iii) R is *symmetric* if $\forall x, y \in S\ (R(x, y) \Rightarrow R(y, x))$.

(iv) R is *antisymmetric* if $\forall x, y \in S\ (R(x, y) \wedge R(y, x) \Rightarrow x = y)$.

(v) R is *transitive* if $\forall x, y, z \in S\ (R(x, y) \wedge R(y, z) \Rightarrow R(x, z))$.

(vi) R is an *equivalence relation* if R is reflexive, symmetric and transitive. In that case, the set $[x]_R = \{y \in S \mid R(x, y)\}$ is called the *equivalence class modulo R* of x, for every $x \in S$. The set of all equivalence classes modulo R is called the *quotient modulo R* of S, denoted by S/R.

(vii) R is *well-founded* if there is no infinite R-descending sequence in S.

(viii) R is a *strict order* if R is irreflexive and transitive.

(ix) R is a *quasi-order*, also called a *preorder*, if R is reflexive and transitive.

(x) R is an *order* if R is reflexive, transitive and antisymmetric. A (strict) order will also be called a (strict) *partial* order, for a clear distinction with the notion of total order defined by the next clause. If $R(x, y) \vee x = y \vee R(y, x)$, then x and y are called R-comparable, or *comparable* for short.

(xi) A (strict) order R is *total* if $\forall x, y \in S\ (R(x, y) \vee x = y \vee R(y, x))$, that is, if every two elements are comparable. A total (strict) order is also called a (strict) *linear* order.

(xii) R is a *well-founded order* if R is a well-founded strict partial order.

(xiii) R is a *well-order* if R is a well-founded strict total order.

Most of the above properties of relations are well-known and evidently useful. The use of a notion like 'well-founded' in a book about term rewriting systems

deserves some explanation. Often rewriting, or even computing in general, can be seen as a process of simplification. For example, 2^{1+2} can be simplified first to 2^3 and then to 8. This induces a relation on expressions, often an order, which decreases under the rewriting process. This relation is well-founded if and only if the rewriting process generally terminates. Moreover, the use of well-founded induction, Theorem A.1.7, is important throughout this book.

A.1.6. DEFINITION. Let S be a set and R a relation on S. A property $\phi(x)$ is *R-inductive* on S if $\forall x \in S\ (\forall y \in S\ (R(y,x) \Rightarrow \phi(y)) \Rightarrow \phi(x))$.

A.1.7. THEOREM. *Let S be a set and R a well-founded relation on S. If $\phi(x)$ is R-inductive on S, then $\phi(x)$ holds everywhere on S.*

PROOF. Let conditions be as above, and assume there exists $x \in S$ such that $\neg\phi(x)$. Since $\phi(x)$ is R-inductive, there exists $y \in S$ such that $R(y,x)$ and $\neg\phi(y)$. Iterating this argument (and using the Axiom of Choice) yields an infinite R-descending sequence. This conflicts with the well-foundedness of R. Now the theorem follows by contraposition. □

A.1.8. EXERCISE. Let S be a set and R a relation on S. For $A \subseteq S$, we say that $x \in A$ is R-minimal if $\neg\exists y \in A\ R(y,x)$. Prove that R is well-founded if and only if every non-empty $A \subseteq S$ contains an R-minimal element.

A.1.9. EXERCISE. Let S be a set and $R \subseteq S \times S$ a relation on S.

(i) Define R' by $R'(x,y) \Leftrightarrow R(x,y) \vee x = y$. Show that R' is a linear order if R is a strict linear order.

(ii) Define R' by $R'(x,y) \Leftrightarrow R(x,y) \wedge x \neq y$. Show that R' is a strict linear order if R is a linear order.

(iii) Define R' by $R'(x,y) \Leftrightarrow R(x,y) \wedge \neg R(y,x)$. Show that R' is a strict partial order if R is a quasi-order.

(iv) Define $\sim$ by $x \sim y \Leftrightarrow R(x,y) \wedge R(y,x)$. Show that $\sim$ is an equivalence relation if R is a quasi-order. Furthermore, show that $\preceq$ defined by $[x]_\sim \preceq [y]_\sim \Leftrightarrow R(x,y)$ is a well-defined partial order on $S/\!\sim$.

(v) Define R' by $R'(x,y) \Leftrightarrow R(x,y) \vee R(y,x)$. Show that R' is the smallest symmetric relation containing R.

Note that, according to Definition A.1.5, strict orders (and hence well-founded orders and well-orders) are *not* orders. Exercise A.1.9(i) and (ii) show that the difference between order and strict order is reflexivity; (iii) and (iv) give two different ways to make an order out of a quasi-order: (iii) by restricting the relation and (iv) by restricting the set by identifying elements in the same equivalence class.

We will use symbols like $<$, $>$, $\subset$, $\supset$, $\sqsubset$, $\sqsupset$, $\prec$, $\succ$ for *strict* orders, and the corresponding underlined symbols $\leq$, $\geq$, $\subseteq$, $\supseteq$, $\sqsubseteq$, $\sqsupseteq$, $\preceq$, $\succeq$ for (quasi-)orders. The underlining expresses reflexivity. The relations $<$ and $>$ are understood to be each other's inverse relations. Similarly for $\preceq$ and $\succeq$,

and so on. The precise relation between $<$ and $\leq$ is: if the strict order $<$ is given, then $\leq = < \cup =$; if the (quasi-)order $\leq$ is given, then $< = \leq - \geq$. For an order $\leq$ we have $< = \leq - =$.

A.1.10. Definition. Let S be a set and $R \subseteq S \times S$ a relation on S.

(i) The *transitive closure* R^+ of R is the relation $\bigcap\{R' \subseteq S \times S \mid R \subseteq R' \text{ and } R' \text{ transitive}\}$. It is not difficult to see that R^+ contains R and is itself transitive. Equivalently, R^+ is the smallest transitive relation containing R. For an alternative characterization, see Exercise A.1.11.

(ii) R is called *acyclic* if the transitive closure of R is irreflexive.

(iii) The *symmetric closure* $R^{\leftrightarrow}$ of R is the relation $\bigcap\{R' \subseteq S \times S \mid R \subseteq R' \text{ and } R' \text{ symmetric}\}$, the smallest symmetric relation containing R. See also Exercise A.1.9(v).

(iv) The *reflexive closure* $R^{=}$ of R is the relation $\bigcap\{R' \subseteq S \times S \mid R \subseteq R' \text{ and } R' \text{ reflexive}\}$. An easy alternative characterization of the reflexive closure of a relation R is given by Exercise A.1.9(i), namely $R^{=}$ equals $R \cup =$.

Closure operations can also be combined, for example R^* denotes the *reflexive–transitive closure* of R (in this case the order in which the closure operations are applied is irrelevant). An important example of closure operations where the order does matter is the following definition. See Exercise A.1.12 for the justification and alternative characterizations.

(v) The *equivalence (relation)* $=_R$ *generated by* R is the relation $\bigcap\{R' \subseteq S \times S \mid R \subseteq R' \text{ and } R' \text{ equivalence relation}\}$. It follows from Exercise A.1.12 that $=_R$ is indeed an equivalence relation coinciding with the reflexive–symmetric–transitive closure of R.

A.1.11. Exercise. Let S be a set and $R \subseteq S \times S$ a relation on S.

(i) Show $R^+(x, z)$ if and only if $x\, R\, y_1\, R\, y_2\, R \cdots R\, y_k\, R\, z$ for some $k \geq 0$ and $y_1, \ldots, y_k \in S$. It is understood that $x\, R\, z$ in case $k = 0$.

(ii) Show that $R^+ = \bigcup\{R^n \mid n > 0\}$ and $R^* = R^+ \cup R^0$.

(iii) Show that R^+ is a well-founded order if and only if R is well-founded.

A.1.12. Exercise. Let S be a set, $R \subseteq S \times S$ a relation on S and let $=_R$ be the equivalence relation generated by R.

(i) Show that $\bigcap E$ is an equivalence relation on S, whenever E is a set of equivalence relations on S.

(ii) Prove or disprove that $=_R$ equals $(R^*)^{\leftrightarrow}$.

(iii) Prove or disprove that $=_R$ equals $(R^{\leftrightarrow})^*$.

(iv) Show $x =_R z$ if and only if $x = y_0\, R^{\leftrightarrow} \cdots R^{\leftrightarrow}\, y_k = z$ for some $k \geq 0$ and $y_0, \ldots, y_k \in S$. It is understood that $x = z$ in case $k = 0$.

A.1.13. Definition. Let $\sqsubseteq$ be a partial order on a set S and $A \subseteq S$.

(i) $x \in A$ is a *minimal element* of A if $\neg\exists y \in A\ y \sqsubset x$.

(ii) $x \in A$ is a *maximal element* of A if $\neg\exists y \in A\ x \sqsubset y$.

(iii) $x \in S$ is a *lower bound* of (for) A if $\forall y \in A\ x \sqsubseteq y$.

(iv) $x \in S$ is an *upper bound*of (for) A if $\forall y \in A\ y \sqsubseteq x$.

(v) If a lower bound x of A is also an element of A, then x is called the *least element* of A, or the *minimum* of A. Least elements are unique, since the order is antisymmetric. Moreover, the minimum is minimal in the sense above, since the strict order is irreflexive.

(vi) If an upper bound x of A is also an element of A, then x is called the *greatest element* of A, or the *maximum* of A. Greatest elements are unique and maximal for similar reasons as above.

(vii) $x \in S$ is an *infimum* of (for) A if x is a greatest lower bound of A, that is, x is the maximum of the set $\{y \in S \mid y \text{ is a lower bound for } A\}$. If this infimum exists, then it is unique and is denoted by either $\mathit{inf}(A)$, or $\mathit{glb}(A)$.

(viii) $x \in S$ is a *supremum* of (for) A if x is a least upper bound of A, that is, x is the minimum of the set $\{y \in S \mid y \text{ is an upper bound for } A\}$. If this supremum exists, then it is unique and is denoted by either $\mathit{sup}(A)$, or $\mathit{lub}(A)$.

All the notions above can be defined for $\sqsubseteq$ being a quasi-order. Then maxima and minima are only unique up to $\sim$ as defined in Exercise A.1.9(iv). The notions of minimal and maximal elements can be generalized to an arbitrary relation.

A.1.14. Definition. Let $\leq$ be a partial order on a set A, $\sqsubseteq$ a partial order on B and let $f : A \to B$.

(i) We say f is *weakly increasing* if $\forall a, a' \in A\ (a \leq a' \Rightarrow f(a) \sqsubseteq f(a'))$.

(ii) We say f is *increasing* if $\forall a, a' \in A\ (a < a' \Rightarrow f(a) \sqsubset f(a'))$.

(iii) We say f is *weakly decreasing* if $\forall a, a' \in A\ (a \leq a' \Rightarrow f(a') \sqsubseteq f(a))$.

(iv) We say f is *decreasing* if $\forall a, a' \in A\ (a < a' \Rightarrow f(a') \sqsubset f(a))$.

Similar terminology is used when starting with strict orders $<$ on A and $\sqsubset$ on B.

Weakly increasing (weakly decreasing) is also called *monotone* (*antitone*). Increasing (decreasing) is also called *strictly monotone (strictly antitone)*. If the order is total, then weakly increasing (weakly decreasing) is also called non-decreasing (non-increasing), since $\neg R(y, x) \wedge x \neq y \Rightarrow R(x, y)$ if R is total.

The following definition gives two ways to combine two strict orders on sets into one on the product set, one pointwise and the other lexicographically, the difference being that the latter assigns priority to the first element of a pair.

A.1.15. Definition. Let $\prec$ and $\prec'$ be strict partial orders on a set S and a set S', respectively. The *product* of $\prec$ and $\prec'$ is the relation $\prec \times \prec'$ on $S \times S'$

defined by $(s_1, s_1') \prec\times\prec' (s_2, s_2')$ if and only if both $s_1 \prec s_2$ and $s_1' \prec' s_2'$. The *lexicographic product* of $\prec$ and $\prec'$ is the relation $\prec\times_{\mathcal{L}}\prec'$ on $S \times S'$ defined by $(s_1, s_1') \prec\times_{\mathcal{L}}\prec' (s_2, s_2')$ if and only if either $s_1 \prec s_2$, or $s_1 = s_2$ and $s_1' \prec' s_2'$.

A.1.16. LEMMA. *Let $\prec$ and $\prec'$ be strict partial orders on a set S and a set S', respectively.*

(i) *Then $\prec\times\prec'$ and $\prec\times_{\mathcal{L}}\prec'$ are strict partial orders on $S \times S'$.*

(ii) *If $\prec$ and $\prec'$ are total, then $\prec\times\prec'$ is not necessarily total.*

(iii) *If $\prec$ and $\prec'$ are total, then $\prec\times_{\mathcal{L}}\prec'$ is total.*

(iv) *If $\prec$ and $\prec'$ are well-founded, then $\prec\times\prec'$ and $\prec\times_{\mathcal{L}}\prec'$ are.*

PROOF. (i) Both irreflexivity and transitivity are obvious.

(ii) Take $S = S' = \mathbb{N}$ with the usual strict total order $<$. Then $(0, 1)$ and $(1, 0)$ are not related by $<\times<$.

(iii) If $\prec$ and $\prec'$ are total, then $\prec\times_{\mathcal{L}}\prec'$ is obviously total too.

(iv) Let $\prec$ and $\prec'$ be well-founded, then $\prec\times\prec'$ is obviously well-founded. Regarding $\prec\times_{\mathcal{L}}\prec'$, assume by contradiction that $(s_0, s_0'), (s_1, s_1'), \ldots$ is an infinite $(\prec\times_{\mathcal{L}}\prec')$-descending sequence. Since $\prec'$ is well-founded, the sequence $s_0, s_1, \ldots$ cannot be constant and there must be a least i such that $s_{i+1} \prec s_i$. Again since $\prec'$ is well-founded, the sequence $s_{i+1}, s_{i+2}, \ldots$ cannot be constant and there must be a least $j > i$ such that $s_{j+1} \prec s_j = s_{i+1}$. Iterating this argument yields an infinite $\prec$-descending subsequence of $s_0, s_1, \ldots$, contradiction. □

A.1.17. DEFINITION. Let $\prec$ be a strict partial order on a set S. The *lexicographic extension* of $\prec$ to the set S^n is the relation $\prec_{\mathcal{L}}$ defined by $\langle s_1, \ldots, s_n\rangle \prec_{\mathcal{L}} \langle t_1, \ldots, t_n\rangle$ if and only if for some $0 \leq i < n$ we have $\langle s_1, \ldots, s_i\rangle = \langle t_1, \ldots, t_i\rangle$ and $s_{i+1} \prec t_{i+1}$. The following lemma states that $\prec_{\mathcal{L}}$ is indeed an order.

A.1.18. LEMMA. *Let $\prec$ be a strict partial order on a set S. Then $\prec_{\mathcal{L}}$ is a strict partial order on the set S^n. Moreover, $\prec_{\mathcal{L}}$ is total (well-founded) if $\prec$ is total (well-founded).*

PROOF. Analogous to the previous proof. □

Totality, or linearity, in the above lemma is a special case of the following exercise treating the more general case of the lexicographic order on finite sequences of arbitrary length. In contrast, well-foundedness does *not* generalize so easily.

A.1.19. EXERCISE. Let $\prec$ be a strict partial order on a set S. The *lexicographic extension* of $\prec$ to S^* is defined by $\langle s_1, \ldots, s_n\rangle \prec_{\mathcal{L}} \langle t_1, \ldots, t_m\rangle$ if and only if for some $0 \leq i \leq n, m$ we have $\langle s_1, \ldots, s_i\rangle = \langle t_1, \ldots, t_i\rangle$ and either $i = n < m$, or $i < n, m$ and $s_{i+1} \prec t_{i+1}$. Prove that $\prec_{\mathcal{L}}$ is a (linear) order on S^* if $\prec$ is a (linear) order on S. Give a counterexample to: $\prec_{\mathcal{L}}$ is a well-order whenever $\prec$ is.

A.1.20. LEMMA. *Let $\prec, \prec', \prec_0, \prec_1, \ldots$ be strict partial orders on a set S and assume $\prec_0 \subseteq \prec_1 \subseteq \cdots$. We consider lexicographic extensions to S^n for some fixed $n \in \mathbb{N}$.*

(i) *If $\prec \subset \prec'$, then $\prec_{\pounds} \subset \prec'_{\pounds}$.*

(ii) $(\bigcup\{\prec_m | \ m \in \mathbb{N}\})_{\pounds} = \bigcup\{(\prec_m)_{\pounds} \mid m \in \mathbb{N}\}$.

PROOF. In view of Definition A.1.17, (i) is obvious. Observe that $\bigcup\{\prec_m \mid m \in \mathbb{N}\}$ is indeed also a strict partial order. The half $\supseteq$ of the equality in (ii) follows by (i). Regarding $\subseteq$, assume $\langle s_1, \ldots, s_n\rangle$ and $\langle t_1, \ldots, t_n\rangle$ are related by $(\bigcup\{\prec_m | \ m \in \mathbb{N}\})_{\pounds}$. Then there exists an i such that $\langle s_1, \ldots, s_i\rangle = \langle t_1, \ldots, t_i\rangle$ and s_{i+1} and t_{i+1} are related by $\bigcup\{\prec_m | \ m \in \mathbb{N}\}$. Hence $s_{i+1} \prec_m t_{i+1}$ for some $m \in \mathbb{N}$, so $\langle s_1, \ldots, s_n\rangle$ and $\langle t_1, \ldots, t_n\rangle$ are related by $(\prec_m)_{\pounds}$, and hence by $\bigcup\{(\prec_m)_{\pounds} \mid m \in \mathbb{N}\}$. □

A.1.21. DEFINITION. Let R be a transitive relation on a set S. A *cone* (with respect to R) is a set $X \subseteq S$ such that $\forall x \in X \ \forall y \in S \ (R(x, y) \Rightarrow y \in X)$. If $X \subseteq S$, then the *cone generated by* X, denoted by $\widehat{X}$, is the set $\{y \in S \mid \exists x \in X \ R(x, y)\}$. A cone X is *finitely generated* if $X = \widehat{Y}$ for some finite $Y \subseteq S$. The cone generated by one element $x \in S$ will be denoted by $\widehat{x}$.

A.1.22. DEFINITION. Let S be a set, $R \subseteq S \times S$ a relation on S and $X \subseteq S$.

(i) X is a *chain* if $\forall x, y \in X \ (R(x, y) \vee x = y \vee R(y, x))$, that is, all elements in X are R-comparable.

(ii) X is an *anti-chain* if $\forall x, y \in X \ (x \neq y \Rightarrow \neg R(x, y) \wedge \neg R(y, x))$, that is, no two different elements of X are comparable.

A.1.23. DEFINITION. A *tree* is a triple $T = (S, \leq, r)$. Here S is a set, elements of S are called *nodes* of T. Furthermore, $\leq$ is a partial order on S and r a distinguished element of S, called the *root* of T, such that the following three conditions are satisfied.

(i) $\forall x \in S \ x \leq r$. This expresses that the root is the greatest element of the tree.

(ii) Also, $\widehat{x}$ is finite for all $x \in S$. This expresses that for every $x \in S$ there are only finitely many elements above x.

(iii) $\forall x, y, z \in S \ (x \leq y \wedge x \leq z \Rightarrow y \leq z \vee z \leq y)$. This clause has various readings, e.g. every cone $\widehat{x}$ is a chain.

The most common reading of (iii) in combination with (ii) is that from every $x \in S$ there is a unique and finite path to the root.[1]

A *path* in the tree T is a chain $C \subseteq S$ such that $\forall x, z \in C \ \forall y \in S \ (x \leq y \leq z \Rightarrow y \in C)$. For example, $\widehat{x}$ is a path for every $x \in S$. In that case

[1]There exists a more general notion of tree in the literature, where paths to the root are not required to be finite, but just well-ordered by $<$. Since we do not need this we chose the simpler notion.

$\widehat{x} = \{x_1, \ldots, x_k\}$ for some $k > 0$ and $x_1, \ldots, x_k \in S$ such that $x = x_1 < \cdots < x_k = r$. This is the path from x to the root, and the length of this path is k. Paths may be finite or infinite, in the latter case the path cannot be of the form $\widehat{x}$ (see the example below). A *maximal* path is a path which is not contained in any other path.

If $x = x_1 < \cdots < x_k = r$ is the path from x to the root and $k > 1$, then x_2 is called the *successor* of x_1. In that case x_1 is called a *predecessor* of x_2. Successors are unique, predecessors are in general not unique. A successor is also called a *father (mother) node*, with the predecessors as *sons (daughters)*. Note that the terminology of successors and predecessors is correct with respect to the direction of the order (the root is the greatest element), but contrary to the intuition that trees grow upward and that children spring out of parents (*descendants* suit better). Minimal nodes are also called *leaves*.

The tree T is called *finitely branching* if every node in the tree has at most finitely many predecessors.

The tree T is *well-founded* if the strict order $<$ is well-founded on S, or, in other words, if all paths are finite.

For all $s \in S$ the tree $T_s = (\{x \in S \mid x \leq s\}, \leq, s)$ is called the *subtree below* s. Indeed, $\leq$ restricted to $\{x \in S \mid x \leq s\}$ is a partial order with greatest element s, and the conditions (ii) and (iii) are inherited from T.

A.1.24. Example. Let S be the set of finite descending sequences of natural numbers. A typical element of S is $\langle n_0, \ldots, n_{k-1} \rangle$ with $k \geq 0$ and $n_0 > \cdots > n_{k-1}$. As partial order on S we take the relation $\succeq$ defined by $x \succeq y$ if and only if y is an initial segment of x. Obviously, $\succeq$ is reflexive, transitive and antisymmetric. As root we take the empty sequence, $\langle\rangle$, indeed the greatest element of S in the order $\succeq$. The tree $T = (S, \succeq, \langle\rangle)$ is graphically represented in Figure A.1, where the lines connect successors (up) to predecessors (down). We have that T is well-founded, but not finitely branching. The subtrees $T_{\langle n \rangle}$ below $\langle n \rangle$ are all finitely branching.

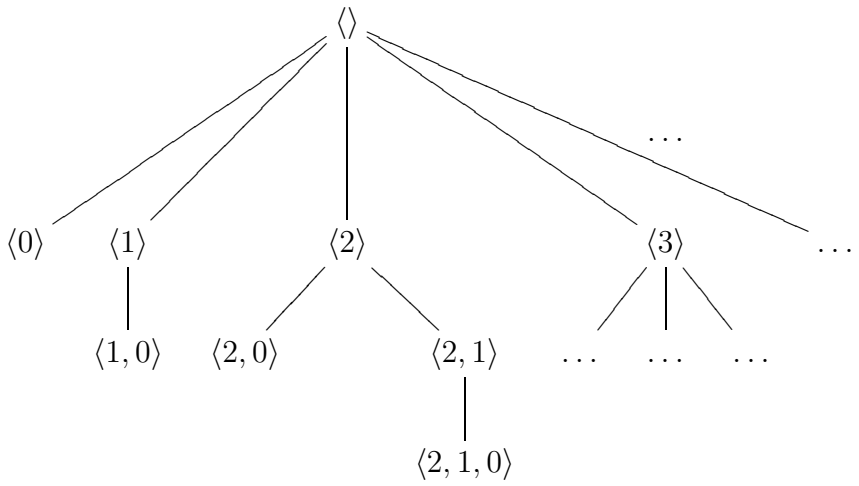


Figure A.1: Example of a well-founded tree

A.1.25. LEMMA (König). *Let $T = (S, \leq, r)$ be a tree. If T is finitely branching and every path in T is finite, then S is finite.*

PROOF. If every path is finite, then the strict order $<$ is well-founded. Now the lemma easily follows by well-founded induction (Theorem A.1.7). □

Without proof we state two well-known equivalents of the Axiom of Choice.

A.1.26. LEMMA (Zorn). *Let $\leq$ be a partial order on a set S. If every chain in S has an upper (lower) bound in S, then S contains a maximal (minimal) element.*

A.1.27. THEOREM (Zermelo). *Every set can be well-ordered, that is, for every set S there exists a well-order $\prec$ on S.*

A.2. Multisets

Multisets are like sets, but with multiple occurrences of elements. Given a set S, subsets have characteristic functions $S \to \{0, 1\}$, whereas multisets are characterized by functions $S \to \mathbb{N}$. In particular, multiset union is additive and multiset difference is subtractive with respect to the multiplicities of the elements (see Equations (A.1) and (A.2) below). For example, the idempotency law $X \cup X = X$ for sets does *not* hold for multisets. For our purpose it suffices to restrict attention to *finite* multisets, that is, multisets with finitely many occurrences of finitely many elements.[2] Whenever we speak of multisets we mean *finite* multisets. Multisets play a role in confluence proofs in Chapter 1, in proving the termination of term rewriting systems, Chapter 6, and in an abstract formulation of completion, Chapter 7.

A.2.1. DEFINITION. Let S be a set. A *multiset* M with elements from S is a function $M : S \to \mathbb{N}$ such that $\{s \in S \mid M(s) > 0\}$, the *set of elements* of M, is finite. Such M is also called a multiset *over* S and can be described explicitly by

$$M = [\underbrace{s_1, \ldots, s_1}_{n_1}, \ \ldots, \underbrace{s_k, \ldots, s_k}_{n_k}]$$

where the $n_i = M(s_i)$ $(1 \leq i \leq k)$ give the *multiplicities* of the elements. In this notation it is tacitly assumed that there are no other elements of M than those explicitly shown. Moreover, permutations of the occurrences of elements in the multiset are allowed and we will often leave the multiplicities implicit, or will express them by using M as function.

The set of multisets over S will be denoted by $S^{\#}$. We define *membership* for multisets by $s \in_{\#} M \Leftrightarrow M(s) > 0$. Multiset *inclusion* is defined by $M \subseteq_{\#} M'$ if and only if $M(s) \leq M'(s)$ for all $s \in S$. As usual, $\subset_{\#}$ is the

[2] More general notions of multisets can be found in the literature.

strict version of $\subseteq_\#$. The *size* $|M|$ of a multiset M is the natural number defined by $|M| = \sum_{s \in S} M(s)$.

Let M and M' be multisets over S. We shall now define the *union*, *difference*, and *intersection* of the multisets M and M'. Let $n \ominus m = n - m$ if $n \geq m$ and 0 otherwise $(n, m \in \mathbb{N})$. Define

$$(M \uplus_\# M')(s) = M(s) + M'(s) \tag{A.1}$$
$$(M -_\# M')(s) = M(s) \ominus M'(s) \tag{A.2}$$
$$(M \cap_\# M')(s) = min\{M(s), M'(s)\} \tag{A.3}$$

Multiset union and intersection are associative and commutative.

A.2.2. Definition. Let $\prec$ be a strict partial order on a set S. We will extend $\prec$ to a strict partial order $\prec_\#$ on $S^\#$, the set of multisets over S, as follows: $\prec_\#$ is the smallest transitive relation satisfying

$$\text{if } \forall x \in_\# M'\ x \prec s, \text{ then } M \uplus_\# M' \ \prec_\# \ M \uplus_\# [s] \tag{A.4}$$

for all $s \in S$, and $M, M' \in S^\#$. The intuition is that a multiset becomes smaller in the sense of $\prec_\#$ by replacing one or more of its elements by an arbitrary number of smaller elements. In particular we can have $M' = [\,]$, in which case the element s is simply deleted. It remains to prove that $\prec_\#$ is indeed a strict partial order. Since $\prec_\#$ is transitive by definition, it suffices to show that $\prec_\#$ is irreflexive. Using Exercise A.1.11, this reduces to showing that the relation defined by (A.4) is acyclic, that is, for all multisets $M_1, \ldots, M_n \in S^\#$ such that $M_n \prec_\# \cdots \prec_\# M_2 \prec_\# M_1$, where all the $\prec_\#$s satisfy (A.4), we have $M_1 \neq M_n$. This can be seen as follows: starting from M_1 we get M_2 by replacing some element s of M_1 by finitely many smaller elements, so M_2 contains one occurrence of s less than M_1. We can only compensate for this missing occurrence of s in M_2 by replacing later on an s' with $s \prec s'$ by a multiset in which s occurs. But then we are missing an even more precious element of M_1, namely s', or even s'' with $s' \prec s''$ in case s' results from replacing s'', etc. It follows that $M_1 \neq M_n$, since $\prec$ on S is acyclic. The reflexive closure of $\prec_\#$ will be denoted by $\preceq_\#$ (and not by $\underline{\prec_\#}$). An alternative characterization of $\preceq_\#$ is given in part (ii) of the following exercise.

A.2.3. Exercise. Let $\prec$ be a strict partial order on a set S. Prove:

(i) For all $M, M' \in S^\#$ we have $M \subseteq_\# M' \ \Leftrightarrow \ M' = (M' -_\# M) \uplus_\# M$.

(ii) Let $M, M' \in S^\#$ and $C = M \cap_\# M'$. Then we have $M \preceq_\# M' \ \Leftrightarrow \forall x \in_\# M -_\# C\ \exists y \in_\# M' -_\# C\ x \prec y$.

(iii) Cancellation for multisets: for all $X, Y, Z \in S^\#$ we have $X \uplus_\# Y \prec_\# X \uplus_\# Z \ \Leftrightarrow \ Y \prec_\# Z$ (also for $\preceq_\#$).

A.2.4. Lemma. *Let $\prec, \prec', \prec_0, \prec_1, \ldots$ be strict partial orders on a set S and assume $\prec_0 \subseteq \prec_1 \subseteq \cdots$.*

(i) *If* $\prec \subset \prec'$, *then* $\prec_\# \subset \prec'_\#$.
(ii) $(\bigcup\{\prec_m \mid m \in \mathbb{N}\})_\# = \bigcup\{(\prec_m)_\# \mid m \in \mathbb{N}\}$.

PROOF. In view of Definition A.2.2, (i) is obvious. Observe that $\bigcup\{\prec_m \mid m \in \mathbb{N}\}$ is indeed also a strict partial order. The half $\supseteq$ of the equality in (ii) follows by (i). Regarding $\subseteq$, assume that the multisets M and M' are related by $(\bigcup\{\prec_m \mid m \in \mathbb{N}\})_\#$. Let $C = M \cap_\# M'$. Using Exercise A.2.3(ii), the finiteness of multisets and the assumption that $\prec_0 \subseteq \prec_1 \subseteq \cdots$, there exists an m such that for all $x \in_\# M -_\# C$ there exist a $y \in_\# M' -_\# C$ such that $x \prec_m y$. Hence $M\ (\prec_m)_\#\ M'$. We conclude that M and M' are related by $\bigcup\{(\prec_m)_\# \mid m \in \mathbb{N}\}$. □

A.2.5. THEOREM. *Let* S *be a set and* $\prec$ *a well-founded order on* S. *Then* $\prec_\#$ *is a well-founded order on* $S^\#$.

PROOF. Let conditions be as above. We will reason by contradiction: let $\cdots \prec_\# M_2 \prec_\# M_1$ be an infinite $\prec_\#$-descending sequence. Observe that $\prec_\#$ is the transitive closure of the relation defined by (A.4) in Definition A.2.2. Using Exercise A.1.11, we can assume without loss of generality that all $\prec_\#$s satisfy (A.4). With the $\prec_\#$-descending sequence $\cdots \prec_\# M_2 \prec_\# M_1$ we can then associate a *labelled tree*. The root of this tree is some designated object *root*. The tree is constructed in stages. At stage 1 we assign to *root* sons labelled with all the elements of M_1 (including duplicates). At stage $n+1$, the tree is constructed from the tree at stage n as follows. If M_{n+1} is obtained from M_n by replacing an element s of M_n by smaller elements $s_1, \ldots, s_k$, possibly containing duplicates, then we assign to a leaf labelled with s sons labelled with $s_1, \ldots, s_k$. Here the particular occurrence of the leaf labelled s doesn't matter. In the case that s is deleted ($k = 0$), we assign one son labelled *deleted* to the leaf labelled s. At any stage n, the multiset M_n can be recovered by collecting the labels of the leaves of the tree that have not been deleted. For example, if $M_1 = [s_1, s_1, s_2]$, $M_2 = [s_1, s'_1, \ldots, s'_k, s_2]$, and $M_3 = [s_1, s'_1, \ldots, s'_k]$, then the labelled tree can be depicted as in Figure A.2. The tree thus associated with the infinite descending sequence is infinite, though finitely branching. It follows by König's Lemma A.1.25 that this tree must contain an infinite path. This infinite path corresponds to an infinite $\prec$-descending sequence, which conflicts with $\prec$ being well-founded. □

A.2.6. EXERCISE. The set A of *amoebae* is inductively defined as follows: if $x_1, \ldots, x_n$ are amoebae, then the collection of $x_1, \ldots, x_n$ within a new membrane is also an amoeba. In the case $n = 0$ this definition gives us ○, the *ur*-amoeba. The x_is ($1 \leq i \leq n$) are the 'sons' of the big amoeba. A *colony* of amoebae is a multiset of amoebae, as in Figure A.3. Amoebae exhibit the following activity: amoeba α may replace simultaneously each of its sons x_i by an arbitrary number of copies $x_i, \ldots, x_i$, say $k_i \geq 0$ copies, of that son; α itself dies in the process, that is, its outer membrane is removed. In particular α may die spontaneously. An example

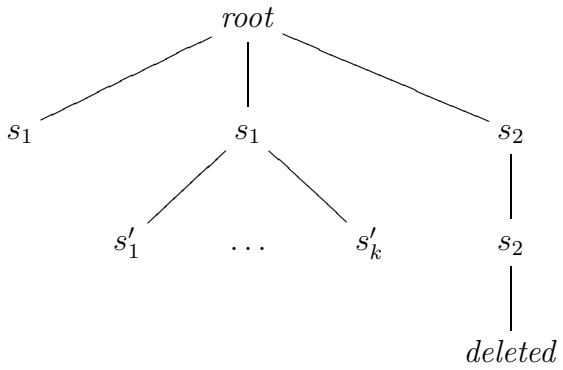


Figure A.2: Labelled tree for $M_3 \prec_\# M_2 \prec_\# M_1$

of a sequence of such steps (a 'life') of an amoebae colony is given in Figure A.3. Prove that a colony of amoebae has only a finite life.

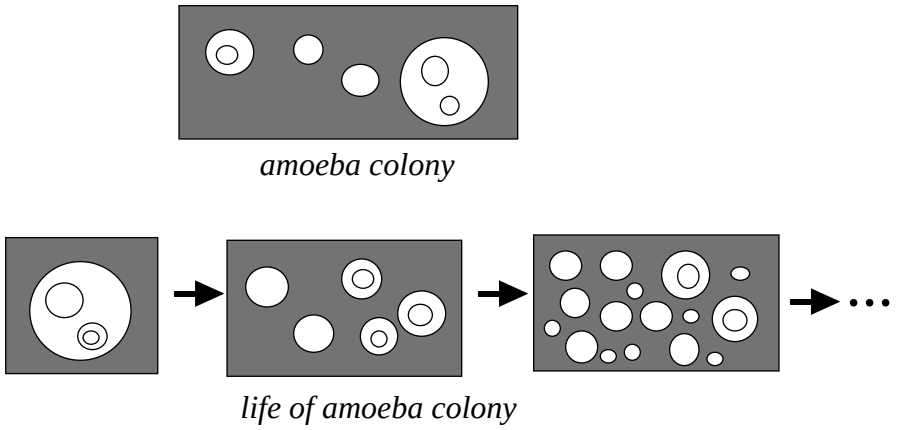


Figure A.3: Amoebae

Next, we extend the capabilities of amoebae by allowing them to reproduce. Two amoebae which can touch each other may reproduce, thereby sharing their outer membrane, and making arbitrarily many copies of their sons (as suggested in Figure A.4). In particular, an amoeba is allowed to multiply its sons and retain its outer membrane, while 'eating' another amoeba. Show that even together with this second rule of activity, each colony must eventually terminate.

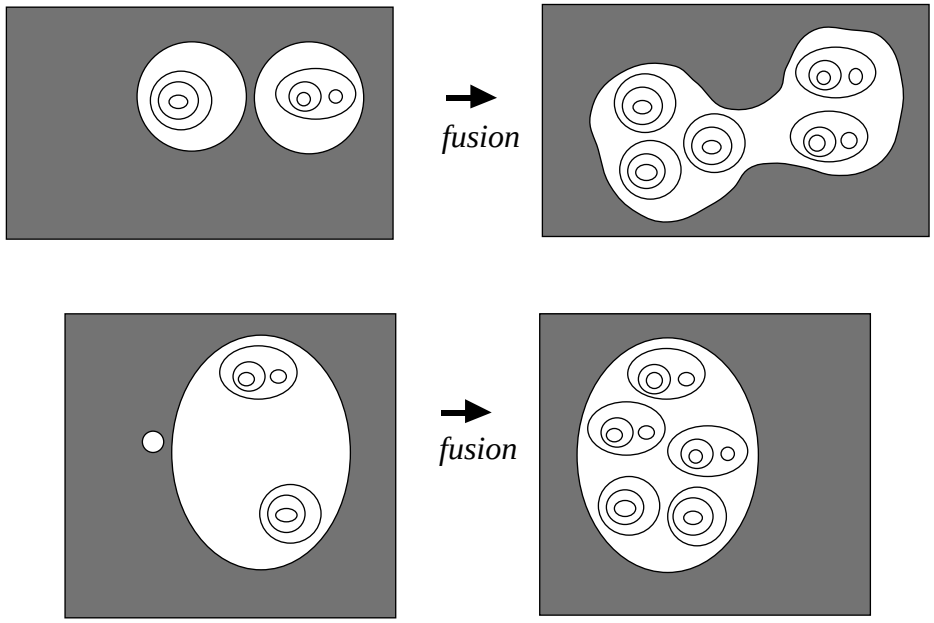


Figure A.4: Fusion of two amoebae

References

Aczel, P. [1978]. A general Church–Rosser theorem, unpublished note, University of Manchester.

Apt, K.R. [1990]. Logic programming, *in:* van Leeuwen [1990], pp. 495–574.

Apt, K.R. [1997]. *From Logic Programming to Prolog*, Prentice-Hall.

Arts, T. [1997]. *Automatically Proving Termination and Innermost Normalisation of Term Rewriting Systems*, Dissertation, Utrecht University.

Avenhaus, J. [1995]. *Reduktionssysteme*, Springer-Verlag.

Baader, F. and T. Nipkow [1998]. *Term Rewriting and All That*, Cambridge University Press.

Bachmair, L. and N. Dershowitz [1986]. Commutation, transformation, and termination, *in:* J.H. Siekmann (ed.), *Proceedings of the Eighth International Conference on Automated Deduction (CADE '86)*, Lecture Notes in Computer Science 230, Springer-Verlag, pp. 5–20.

Bachmair, L. and D. Plaisted [1985]. Associative path orderings, *in:* J.-P. Jouannaud (ed.), *Proceedings of the First International Conference on Rewriting Techniques and Applications (RTA '85)*, Lecture Notes in Computer Science 202, Springer-Verlag, pp. 241–254.

Barendregt, H.P. [1984]. *The Lambda Calculus: Its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics 103, 2nd revised edition, North-Holland.

Barendregt, H.P. [1992]. Lambda calculi with types, *in:* S. Abramsky, D.M. Gabbay and T.S.E. Maibaum (eds.), *Handbook of Logic in Computer Science*, Vol. 2, Oxford University Press, pp. 117–310.

Ben-Cherifa, A. and P. Lescanne [1987]. Termination of rewriting systems by polynomial interpretations and its implementation, *Science of Computing Programming* **9**(2), pp. 137–159.

Bergstra, J.A. and J.W. Klop [1986]. Conditional rewrite rules: confluence and termination, *Journal of Computer and System Sciences* **32**, pp. 323–362.

Bertling, H. and H. Ganzinger [1989]. Completion-time optimization of rewrite-time goal solving, *in:* N. Dershowitz (ed.), *Proceedings of the Third Conference on Rewriting Techniques and Applications (RTA '89)*, Lecture Notes in Computer Science 355, Springer-Verlag, pp. 45–58.

Birkhoff, G. [1935]. On the structure of abstract algebras, *Proceedings of the Cambridge Philosophical Society* **31**(4), pp. 433–454.

Book, R.V. [1987]. Thue systems as rewriting systems, *Journal of Symbolic Computation* **3**, pp. 39–68.

Church, A. [1936]. An unsolvable problem of elementary number theory, *Annals of Mathematics* **33**, pp. 346–366.

Church, A. [1941]. *The Calculi of Lambda-Conversion*, Annals of Mathematics Studies 6, Princeton University Press.

Courcelle, B. [1990]. Recursive application schemes, *in:* van Leeuwen [1990], pp. 459–492.

Curry, H.B. [1930]. Grundlagen der kombinatorischen Logik, *American Journal of Mathematics* **52**, pp. 509–536, 789–834.

Curry, H.B. and R. Feys [1958]. *Combinatory Logic*, Vol. I, North-Holland.

van Dalen, D. [1994]. *Logic and Structure*, 3rd revised edition, Springer-Verlag.

Dauchet, M. [1989]. Simulation of Turing machines by a left-linear rewrite rule, *in:* N. Dershowitz (ed.), *Proceedings of the Third International Conference on Rewriting Techniques and Applications (RTA '89)*, Lecture Notes in Computer Science 355, Springer-Verlag, pp. 109–120.

Dauchet, M. and S. Tison [1984]. Decidability of confluence for ground term rewriting systems, Technical report, Université de Lille I.

Dauchet, M., T. Heuillard, P. Lescanne and S. Tison [1990]. Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems, *Information and Computation* **88**(2), pp. 187–201.

Dershowitz, N. [1987]. Termination of rewriting, *Journal of Symbolic Computation* **3**(1), pp. 69–116. Corrigendum in **4**(3), pp. 409–410.

Dershowitz, N. and J.-P. Jouannaud [1990]. Rewrite systems, *in:* van Leeuwen [1990], pp. 243–320.

Dershowitz, N. and M. Okada [1990]. A rationale for conditional equational programming, *Theoretical Computer Science* **75**, pp. 111–138.

Dershowitz, N., J.-P. Jouannaud and J.W. Klop [1993]. More problems in rewriting, *in:* C. Kirchner (ed.), *Proceedings of the Fifth Conference on Rewriting Techniques and Applications (RTA '93)*, Lecture Notes in Computer Science 690, Springer-Verlag, pp. 468–487. Included in the RTA list of open problems at `http://www.lri.fr/~rtaloop`.

Dershowitz, N., M. Okada and G. Sivakumar [1988]. Canonical conditional rewrite systems, *in:* E. Lusk and R. Overbeek (eds.), *Proceedings of the Ninth International Conference on Automated Deduction (CADE '88)*, Lecture Notes in Computer Science 310, Springer-Verlag, pp. 538–549.

Dijkstra, E.W. [1976]. *A Discipline of Programming*, Prentice-Hall.

Doets, K. [1994]. *From Logic to Logic Programming*, MIT Press.

Drosten, K. [1989]. *Termersetzungssysteme*, Informatik-Fachberichte 210, Springer-Verlag.

Ehrig, H. and B. Mahr [1985]. *Fundamentals of Algebraic Specifications 1, Equations and Initial Semantics*, EATCS Monographs on Theoretical Computer Science 6, Springer-Verlag.

Enderton, H.B. [1977]. *Elements of Set Theory*, Academic Press.

Geser, A., A. Middeldorp, E. Ohlebusch and H. Zantema [1997]. Relative undecidability in term rewriting, *in:* D. van Dalen and M. Bezem (eds.), *Tenth International Workshop on Computer Science Logic (CSL '96), Selected Papers*, Lecture Notes in Computer Science 1258, Springer-Verlag.

Hankin, C. [1994]. *Lambda calculi, a guide for computer scientists*, Oxford University Press.

Hanus (ed.), M. [2000]. Curry: An integrated functional logic language (vers. 0.7.1), `http://www.informatik.uni-kiel.de/~mh/curry/report.html`.

Hanus, M. [1997]. A unified computation model for functional and logic programming, *in: Conference Record of the Twenty-Fourth ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '97)*, pp. 80–93.

Hindley, J.R. [1964]. *The Church–Rosser property and a result in combinatory logic*, Dissertation, University of Newcastle-upon-Tyne.

Hindley, J.R. and J.P. Seldin [1986]. *Introduction to Combinators and λ-Calculus*, Cambridge University Press.

Hofstadter, D. [1979]. *Gödel, Escher, Bach: an Eternal Golden Braid*, Basic Books.

Hopcroft, J.E., R. Motwani and J.D. Ullman [2001]. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley.

Huet, G. [1980]. Confluent reductions: abstract properties and applications to term rewriting systems, *Journal of the ACM* **27**(4), pp. 797–821.

Huet, G. [1994]. Residual theory in λ-calculus: a formal development, *Journal of Functional Programming* **4**(3), pp. 371–394.

Huet, G. and D.S. Lankford [1978]. On the uniform halting problem for term rewriting systems, Rapport Laboria 283, IRIA.

Huet, G. and J.-J. Lévy [1991]. Computations in orthogonal rewriting systems, Part I + II, *in:* J.L. Lassez and G.D. Plotkin (eds.), *Computational Logic – Essays in Honor of Alan Robinson*, MIT Press, pp. 395–443. Revised version of: Call-by-need computations in non-ambiguous linear term rewriting systems, Rapport 359, INRIA, 1979.

Jantzen, M. [1988]. *Confluent String Rewriting and Congruences*, EATCS Monographs on Theoretical Computer Science 14, Springer-Verlag.

Jouannaud, J.-P. and H. Kirchner [1986]. Completion of a set of rules modulo a set of equations, *SIAM Journal of Computing* **15**(4), pp. 1155–1194.

Kahrs, S. [1995]. Confluence of curried term-rewriting systems, *Journal of Symbolic Computation* **19**(6), pp. 601–623.

Kennaway, J.R., J.W. Klop, M.R. Sleep and F.J. de Vries [1995]. Transfinite reductions in orthogonal term rewriting systems, *Information and Computation* **119**(1), pp. 18–38.

Khasidashvili, Z. [1990]. Expression Reduction Systems, *Proceedings of I. Vekua Institute of Applied Mathematics* **36**, pp. 200–220.

Khasidashvili, Z. [1993a]. On the equivalence of persistent term rewriting systems and recursive program schemes, *in: Proceedings of the Second Israel Symposium on Theory of Computing and Systems*, IEEE Computer Society Press, pp. 240–249.

Khasidashvili, Z. [1993b]. Optimal normalization in orthogonal term rewriting systems, *in:* C. Kirchner (ed.), *Proceedings of the Fifth International Conference on Rewriting Techniques and Applications (RTA '93)*, Lecture Notes in Computer Science 690, Springer-Verlag, pp. 243–258.

Klop, J.W. [1980]. *Combinatory Reduction Systems*, Mathematical Centre Tracts 127, Mathematisch Centrum.

Klop, J.W. [1985]. Term rewriting systems, Unpublished lecture notes for the seminar on reduction machines, Ustica.

Klop, J.W. [1992]. Term rewriting systems, *in:* S. Abramsky, D.M. Gabbay and T.S.E. Maibaum (eds.), *Handbook of Logic in Computer Science*, Vol. 2, Oxford University Press, pp. 1–116.

Klop, J.W., A. Middeldorp, Y. Toyama and R.C. de Vrijer [1994]. Modularity of confluence: a simplified proof, *Information Processing Letters* **49**, pp. 101–109.

Knuth, D.E. and P.B. Bendix [1970]. Simple word problems in universal algebra, *in:* J. Leech (ed.), *Computational Problems in Abstract Algebra*, Pergamon Press, pp. 263–297.

Lankford, D. S. [1979]. On proving term rewriting systems are noetherian, Technical Report MTP–3, Louisiana Technical University.

van Leeuwen, J. (ed.) [1990]. *Handbook of Theoretical Computer Science, Vol. B, Formal Models and Semantics*, Elsevier.

Lévy, J.-J. [1976]. An algebraic interpretation of the $\lambda\beta K$-calculus, and an application of a labelled λ-calculus, *Theoretical Computer Science* **2**(1), pp. 97–114.

Martelli, A. and U. Montanari [1982]. An efficient unification algorithm, *Transactions on Programming Languages and Systems* **4**(2), pp. 258–282.

Matijasevich, Y. [1967]. Simple examples of undecidable associative calculi, *Soviet Mathematics (Doklady)* **8**, pp. 555–557.

McCarthy, J. [1960]. Recursive functions of symbolic expressions and their computation by machine, *Communications of the ACM* **3**(4), pp. 184–195.

Meinke, K. and J.V. Tucker [1991]. Universal algebra, *in:* S. Abramsky, D. Gabbay and T.S.E. Maibaum (eds.), *Handbook of Logic in Computer Science*, Vol. 1, Oxford University Press, pp. 189–411.

Middeldorp, A. [1989]. A sufficient condition for the termination of the direct sum of term rewriting systems, *in: Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 89)*, IEEE Computer Society Press, pp. 396–401.

Middeldorp, A., H. Ohsaki and H. Zantema [1996]. Transforming termination by self-labelling, *in:* M.A. McRobbie and J.K. Slaney (eds.), *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE '96)*, Lecture Notes in Artificial Intelligence 1104, Springer-Verlag, pp. 373–387.

Nederpelt, R.P. [1973]. *Strong Normalization for a Typed Lambda Calculus with Lambda Structured Types*, Dissertation, Technische Hogeschool Eindhoven.

Newman, M.H.A. [1942]. On theories with a combinatorial definition of 'equivalence', *Annals of Mathematics* **43**, pp. 223–243.

O'Donnell, M.J. [1977]. *Computing in Systems Described by Equations*, Lecture Notes in Computer Science 58, Springer-Verlag.

Ohlebusch, E. [1993]. A simple proof of sufficient conditions for the termination of the disjoint union of term rewriting systems, *Bulletin of the European Association for Theoretical Computer Science* **49**, pp. 178–183.

Oyamaguchi, M. [1987]. The Church–Rosser property for ground term rewriting systems is decidable, *Theoretical Computer Science* **49**(1), pp. 43–79.

Peterson, G.E. and M.E. Stickel [1981]. Complete sets of reductions for some equational theories, *Journal of the ACM* **28**(2), pp. 233–264.

Plaisted, D.A. [1985]. Semantic confluence tests and completion methods, *Information and Control* **65**, pp. 182–215.

Post, E.L. [1946]. A variant of a recursively unsolvable problem, *Bulletin of the American Mathematical Society* **52**, pp. 264–268.

Probst, D. and T. Studer [2001]. How to normalize the jay, *Theoretical Computer Science* **254**(1–2), pp. 677–681.

Rogers, H., Jr. [1967]. *Theory of Recursive Functions and Effective Computability*, McGraw-Hill.

Rosen, B.K. [1973]. Tree-manipulating systems and Church–Rosser theorems, *Journal of the ACM* **20**(1), pp. 160–187.

Rusinowitch, M. [1987]. On termination of the direct sum of term rewriting systems, *Information Processing Letters* **26**, pp. 65–70.

Schmidt-Schauß, M., M. Marchiori and S.E. Panitz [1995]. Modular termination of r-consistent and left-linear term rewriting systems, *Theoretical Computer Science* **149**(2), pp. 361–374.

Schönfinkel, M. [1924]. Über die Bausteine der mathematischen Logik, *Mathematische Annalen* **92**, pp. 305–316.

Scott, D.S. [1963]. A system of functional abstraction, unpublished.

Scott, D.S. [1975]. Combinators and classes, *in:* C. Böhm (ed.), *λ-Calculus and Computer Science Theory*, Lecture Notes in Computer Science 37, Springer-Verlag, pp. 1–26.

Smullyan, R. [1985]. *To Mock a Mockingbird*, Oxford University Press.

Staples, J. [1975]. Church–Rosser theorems for replacement systems, *in:* J. Crossley (ed.), *Algebra and Logic*, Lecture Notes in Mathematics 450, Springer-Verlag, pp. 291–307.

Suzuki, T., A. Middeldorp and T. Ida [1995]. Level-confluence of conditional rewrite systems with extra variables in right-hand sides, *in:* J. Hsiang (ed.), *Proceedings of the Sixth International Conference on Rewriting Techniques and Applications (RTA '95)*, Lecture Notes in Computer Science 914, Springer-Verlag, pp. 179–193.

Terese [2003]. *Term Rewriting Systems*, Cambridge Tracts in Theoretical Computer Science 55, Cambridge University Press.

Toyama, Y. [1987a]. Counterexamples to termination for the direct sum of term rewriting systems, *Information Processing Letters* **25**, pp. 141–143.

Toyama, Y. [1987b]. On the Church–Rosser property for the direct sum of term rewriting systems, *Journal of the ACM* **34**(1), pp. 128–143.

Toyama, Y., J.W. Klop and H.P. Barendregt [1989]. Termination for the direct sum of left-linear term rewriting systems, *in:* N. Dershowitz (ed.), *Proceedings of the Third International Conference on Rewriting Techniques and Applications (RTA '89)*, Lecture Notes in Computer Science 355, Springer-Verlag, pp. 477–491.

Toyama, Y., J.W. Klop and H.P. Barendregt [1995]. Termination for direct sums of left-linear complete term rewriting systems, *Journal of the ACM* **42**(6), pp. 1275–1304.

Turner, D.A. [1979]. A new implementation technique for applicative languages, *Software Practice and Experience* **9**, pp. 31–49.

Turner, D.A. [1986]. An overview of miranda, *SIGPLAN Notices* **21**(12), pp. 158–166. Also appeared as Turner [1990].

Turner, D.A. [1990]. An overview of miranda, *in:* D.A. Turner (ed.), *Research Topics in Functional Programming*, Addison-Wesley, pp. 1–16.

Waldmann, J. [2000]. The combinator **S**, *Information and Computation* **159**(1–2), pp. 2–21.

Yamada, Toshiyuki, Jürgen Avenhaus, Carlos Loría-Sáenz and Aart Middeldorp [2000]. Logicality of conditional rewrite systems, *Theoretical Computer Science* **236**(1,2), pp. 209–232.

List of notations

$(A, \{\rightarrow_\alpha | \ \alpha \in I\})$, 7
$(A, \rightarrow)$, 7
$\rightarrow$, 7
$\rightarrow_I$, 7
$\rightarrow_\alpha$, 7
$=_\alpha$, 8
$\leftrightarrow_\alpha$, 8
$\twoheadrightarrow_\alpha$, 8
$\sigma : a \twoheadrightarrow_I b$, 8
$\sigma : a \twoheadrightarrow_\alpha b$, 8
$\rightarrow_{\alpha\beta}$, 8
$\rightarrow_\alpha^+$, 8
$\rightarrow_\alpha^{-1}$, 8
$\rightarrow_\alpha^\equiv$, 8
$a \rightarrow_I b$, 8
$a \rightarrow_\alpha b$, 8
$\twoheadrightarrow_{\alpha\beta}$, 8
$\rightarrow_\alpha \cdot \rightarrow_\beta$, 8
$\mathcal{G}(a)$, 9
$\mathcal{A} \subseteq \mathcal{B}$, 9
$\mathrm{CR}^{\leq 1}$, 10
DP, 10
TP, 10
CR, 10
WCR, 10
SN, 12
WN, 12
$nf(a)$, 13
FB, 13
Inc, 13
Ind, 13
NF, 13
UN, 13
WF, 13
CR^1, 20
Σ, 22
Var, 22
$F(t_1, \ldots, t_n)$, 23
$\Box$, 23
$Ter(\Sigma)$, 23
$Ter_0(\Sigma)$, 23
$\equiv$, 23
$root(t)$, 23
$|t|$, 23
$C[\,]$, 24
$C[t]$, 24
$C[t_1, \ldots, t_n]$, 24
$\langle s \mid C[\,]\rangle$, 24
$s \subset t$, 24
$s \subseteq t$, 24
$depth(t)$, 25
$\langle s' \mid C'[\,]\rangle \leq \langle s \mid C[\,]\rangle$, 25
$s \leq t$, 25
$\langle k_1, \ldots, k_n\rangle$, 26
$p \cdot q$, 26
$p \leq q$, 26
$p \mid\mid q$, 26
$t(p)$, 27
$t[\,]_p$, 27
$t\,|_p$, 27
$[\vec{x} := \vec{s}]$, 28
$[x_1, \ldots, x_n := s_1, \ldots, s_n]$, 28
$Dom(\sigma)$, 28
$\{s_1/x_1, \ldots, s_n/x_n\}$, 28
$\{x_1/s_1, \ldots, x_n/s_n\}$, 28
$\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$, 28
$t\sigma$, 28
t^σ, 28
$s \cong t$, 29
$s \prec t$, 29
$s \preceq t$, 29
$\sigma \preceq \tau$, 30
$\rho\colon l \rightarrow r$, 31
$\rightarrow_\rho$, 31
(Σ, R), 33
$\rightarrow$, 33
$\rightarrow_R$, 33
$t \rightarrow_{C[\,]} s$, 33
$t \rightarrow_{\rho, C[\,]} s$, 33
$\mathcal{R}$, 33
$t \twoheadrightarrow s$, 34
$t = s$, 35
$\mathcal{N}^{AM}$, 35
$\mathcal{N}_=^{AM}$, 37
$E \vdash s = t$, 39
$s =_E t$, 39

$\mathcal{E}^{\leftrightarrow}$, 39
$\mathcal{R}^{=}$, 39
$\mathcal{R}^{\leftrightarrow}$, 39
$F^{\mathcal{A}}$, 40
$\mathcal{A}$, 40
$[\![t]\!]^{\mathcal{A},\theta}$, 40
$\models$, 41
$\langle t, s\rangle$, 49
Ap, 57
I, 57
K, 57
S, 57
$t \cdot s$, 57
ts, 58
$[x]t$, 61
Y, 63
Y_C, 63
CL-e, 65
$(\mathcal{R})_0$, 67
R^{cur}, 67
Σ^{cur}, 67
$cur(t)$, 67
ρ_u, 74
$t \to s \Leftarrow C_1, \ldots, C_n$, 74
$t \downarrow s$, 74
$\mathcal{C}_u$, 74
$\to_n$, 76
$\mathrm{CL} + \delta$, 81
$|t|$, 83
f^l, 83
$\mathcal{R}^L = (\Sigma^L, R^L)$, 83
s_i/s_j, 87
$\twoheadrightarrow$, 89
$\underline{\mathcal{R}} = (\underline{\Sigma}, \underline{R})$, 91
$\underline{f}$, 91
$\underline{\mathrm{CL}}$, 92
$t \geq s$, 93
$s \underline{\twoheadrightarrow} t$, 94
$\underline{\leq}_S$, 96
$\underline{\leq}_s$, 96
t_S, 96
t_s, 96
T/S, 97
$t \multimap t'$, 99
$\varnothing$, 106
ρ/σ, 106
$\infty(t)$, 108
$\mathbb{F}$, 112
$\mathbb{F}_{\mathrm{GK}}$, 113
$\mathbb{F}_{\mathrm{lm}}$, 113
$\mathbb{F}_{\mathrm{po}}$, 113
$\rhd$, 123
$\uplus$, 129
$\mathcal{R}_b \cup \mathcal{R}_w$, 129
$\mathcal{R}_b \uplus \mathcal{R}_w$, 129
$C[\![s_1, \ldots, s_n]\!]$, 130
$S(t)$, 130
$rank(t)$, 130
$\langle s_1, \ldots, s_n\rangle \propto \langle t_1, \ldots, t_n\rangle$, 132
$\to_i$, 132
$\to_o$, 132
Σ_A, 141
f_A, 141
$<_A$, 142
$[\![t]\!]^{\alpha}$, 142
$ax_1^{k_1}x_2^{k_2}$, 145
$\sum_{i=1}^{m} a_i X_1^{k_{i,1}} X_2^{k_{i,2}} \ldots X_n^{k_{i,n}}$, 146
$\geq_A$, 149
$\ll$, 149
(Σ, E), 152
$(\Sigma, E) \vdash s = t$, 152
$F^{Ter(\Sigma)}$, 154
$Ter(\Sigma)$, 154
$Ter_0(\Sigma)$, 154
$F^{\mathcal{A}/\sim}$, 155
$\mathcal{T}(\Sigma, E)$, 155
$\mathcal{T}_0(\Sigma, E)$, 155
$\mathcal{A}/\sim$, 155
$(\Sigma, E) \models s = t$, 156
$\cong$, 158
$I(\Sigma, E)$, 159
$\mathbb{Z}$, 167
$\mathbb{N}$, 167
$\{s \in S \mid \phi(s)\}$, 167
$R{\cdot}R'$, 168
R^n, 168
$\circ$, 168
$\cdot$, composition, 168
$f(X)$, 168
f^n, 168
f^{-1}, 168
$f^{-1}(Y)$, 168
S/R, 169
S^*, 169
S^n, 169
$[x]_R$, 169
$\cdot$, concatenation, 169
$\langle\rangle$, 169
$\langle s_0, s_1, \ldots, s_{n-1}\rangle$, 169
$s_0 s_1 \cdots s_{n-1}$, 169
R^+, 171
R^*, 171
$R^{\leftrightarrow}$, 171
$R^{=}$, 171
$glb(A)$, 172
$inf(A)$, 172

$lub(A)$, 172
$\prec \times \prec'$, 172
$sup(A)$, 172
$\prec \times_{\pounds} \prec'$, 173
$\prec_{\pounds}$, 173
$T = (S, \leq, r)$, 174
$\widehat{X}$,$\widehat{x}$, 174
$T_s = (\{x \in S \mid x \leq s\}, \leq, s)$, 175
$S^{\#}$, 176
$[s_1, \ldots, s_1, \ \ldots, s_k, \ldots, s_k]$, 176
$\in_{\#}$, 176
$\subset_{\#}, \subseteq_{\#}$, 176
$\uplus_{\#}$, 177
$\cap_{\#}$, 177
$\cup_{\#}$, 177
$^-{}_{\#}$, 177
$\prec_{\#}$, 177

Index of authors

Aczel, P., 86
Apt, K.R., 30, 72
Arts, T., 3
Avenhaus, J., x, 70, 77

Baader, F., x, 49, 72
Bachmair, L., 10, 71
Barendregt, H.P., x, 19, 32, 60, 65, 108, 127, 129, 136, 157
Ben-Cherifa, A., 144
Bendix, P.B., ix, 51, 52, 163, 164
Bergstra, J.A., 74
Bertling, H., 39
Birkhoff, G., 156, 157
Book, R.V., 70

Church, A., 22, 81, 110
Courcelle, B., 70
Curry, H.B., 32, 57, 60, 67, 86

Dauchet, M., 121, 122
Dershowitz, N., 10, 12, 49, 72, 74, 77, 122
Dijkstra, E.W., 56
Doets, K., 30
Drosten, K., 70, 135

Ehrig, H., 70, 72

Feys, R., 32, 57, 60, 86

Ganzinger, H., 39
Geser, A., 120
Grabmayer, C., 37

Hamming, R., 56
Hankin, C., 127
Hanus, M., 72
Herbrand, J., 5
Heuillard, T., 121
Hindley, J.R., 18, 19, 60
Hofstadter, D, 71
Hopcroft, J.E., 120
Huet, G., 12, 13, 49, 51, 79, 99, 115, 121, 122

Ida, T., 77

Jantzen, M., 70
Jouannaud, J.-P., 12, 49, 165

Kahrs, S., 67
Kennaway, J.R., 67
Khasidashvili, Z., 70, 111
Kirchner, H., 165
Klop, J.W., 53, 65, 74, 129, 134, 136
Knuth, D.E., ix, 51, 52, 163, 164

Lankford, D.S., 121, 122, 140
Lescanne, P., 121, 144
Lévy, J.-J., 106, 115
Loría-Sáenz, C., 77

Mahr, B., 70, 72
Marchiori, M., 129, 136
Martelli, A., 30
Matijasevich, Y., 120
McCarthy, J., x
Meinke, K., 72
Middeldorp, A., 67, 77, 120, 129, 136
Montanari, U., 30
Motwani, R., 120

Nederpelt, R.P., 17
Newman, M.H.A., 14, 20
Nipkow, T., x, 49, 72

O'Donnell, M.J., 111, 115
Ohlebusch, E., 120, 129, 137
Ohsaki, H., 67
Okada, M., 72, 74, 77
Oyamaguchi, M., 121

Panitz, S.E., 129, 136
Peterson, G.E., 165
Plaisted, D.A., 41, 71
Plump, D., 37
Post, E.L., 71, 126
Probst, D., 108

Rogers, H., Jr., 120
Rosen, B.K., 18, 82
Rusinowitch, M., 129, 136

Schmidt-Schauß, M., 129, 136
Schönfinkel, M., 57, 67
Scott, D.S., 60, 127
Seldin, J.P., 60
Sivakumar, G., 74, 77
Sleep, M.R., 67
Smullyan, R., 60
Staples, J., 19
Stickel, M.E., 165
Studer, T., 108
Suzuki, T., 77

Tison, S., 121
Toyama, Y., 129, 134–136, 151
Tucker, J.V., 72
Turner, D.A., x, 57, 64

Ullman, J.D., 120

Visser, A., 62
de Vries, F.J., 67
de Vrijer, R.C., 129

Waldmann, J., 62, 128

Yamada, T, 77

Zantema, H., 67, 120

Index of subjects

absorption, 108
abstract reduction system, 7, 34, *see also* ARS
abstraction, 60
acyclic, 171
algebra, 40, 141
  initial, 158
  quotient, 155
Σ-algebra, 40
alphabet, 22
ambiguous, 78
amoeba, 178
ancestor, 85
anti-chain, 174
antitone, 172
application, 57
applicative notation, 58, 65
applicative TRS, 57, 66
argument, 23
  of a redex, 45, 48
arity, 22
ARS, 7, 34, *see also* abstract reduction system
  canonical, 13
  complete, 13
  consistent, 20
  semi-complete, 13
  sub-, 9
association to the left, 58

bijection, 168
Birkhoff's Completeness Theorem, 157
bisimulation, 125
Booleans, 42
bound
  greatest lower, 172
  least upper, 172
  lower, upper, 172

chain, 174
Church's δ-rules, 81
Church's Theorem, 110
Church–Rosser, 10
  weakly, 10
CL, 57, 80
class, equivalence -, 169
CL-e, 65
closed fragment, 67
closed term, 23
closure
  equivalence, 171
  reflexive, 171
  reflexive–transitive, 171
  symmetric, 171
  transitive, 171
coefficient, 145
cofinal, 13, 112
cofinality property, 13
coherence, 86
collapsing, 36, 136
Collatz's problem, 2
combinator, 60
  basic, 59
  fixed-point, 63
combinatorial completeness, 60, 61
combinatory logic, 57, 80
  undecidability, 127
commute, 10
  weakly, 10
comparable, wrt. to relation, 169
compatible, 141, 142
complete, 34, 42
  ARS, 13
  semi-, 13
complete development, 96, 97
completely normalizing, 111
completion, 43, 54
composition, 168
concatenation, 169
condition, 73
  negative, 76
conditional rewrite rule, 74
conditional specification, 72
conditional term rewriting system, 74, 75
  generalized, 75

join, 74
normal, 74
semi-equational, 74
cone, 174
finitely generated, 174
configuration, 123
confluence, 2
local, 10
strong, 12
undecidable, 120
weak, 10
confluent, 10, 34
Σ-congruence, 154
conservativity, of ARSs, 20
constant, 22
constant symbol, 22
constructor, 69
constructor TRS, 69
context, 1, 23
one-hole, 24
trivial, 24
contraction, 32
contractum, 31
convergent, 51
conversion, 9
final, 102
initial, 101
intermediate, 102
conversion sequence, 9
convertibility (relation), 7, 8
convertible, 9
CP, 13
CR, 10, 11, 34
CR^1, 12, 20
$\mathrm{CR}^{\leq 1}$, 10, 11
created redex, 86
critical pair, 44, 49
completion, 43, 160
convergent, 51
trivial, 79
critical pair completion, 54
Critical Pair Lemma, 51, 86, 87
critical reduction step, 108
CTRS, 74, 75
Curry, 72
currying, 67

decidability, 119
local, 119
relative, 120
uniform, 119
decidable, 119
decreasing, 172
depth, 25
descendant, 83, 85
destructive at level 2, 137
destructive step, 131
development, 96
diagram
elementary, 100
diamond property, 10, 12
difference
multiset, 177
discriminator, 66
disjoint occurrences, 25
disjoint sum, 129
disjoint union, 129
domain, 40
of a substitution, 28
DP, 10–12
duplicating, 36, 136

e.d., 100
element
R-minimal, 170
greatest, 172
least, 172
maximal, 171
minimal, 171
of multiset, 176
elementary diagram, 87, 100
improper, 101
splitting, 100
elementary step, 1
empty reduction, 98, 106
empty step, 101, 106
equational logic, 152
equational specification, 38, 152
equivalence
generated by, 171
class, 169
permutation, 106
projection, 106
relation, 169
erased, 100
erasing, 36
Erasure Lemma, 109
Eratosthenes, sieve of, 80
expansion
α-, 8
extension, conservative, 20
extension, lexicographic, 173

fair, 115
FB, 13
FD, 97
FD^+, 97

Fibonacci numbers, 56, 77
final conversion, 102
Finite Developments Theorem, 97
finitely branching, 13
finitely branching tree, 175
first-order, 22
fixed point, 63, 168
fixed-point combinator, 63
function
    (strictly) antitone, 172
    (strictly) monotone, 172
    inverse, 168
    bijective, 168
    decreasing, 172
    identity, 168
    increasing, 172
    injective, 168
    partial, 167
    surjective, 168
    total, 167
function symbol, 22
functional notation, 65

generalized CTRS, 75
ground term, 23
ground TRS, 67, 121
ground-CR
    decidable, 121
ground-SN
    decidable, 121
ground-complete, 35, 68
ground-CR, 35, 68
ground-SN, 35, 68
ground-WN, 35, 68
group, 50, 163

halting problem, 120
Hamming number, 56
head symbol, 23
higher-order, 22
hole, 23
Σ-homomorphism, 158

identity function, 168
image, (inverse), 168
Inc, 13
inclusion, for multisets, 176
increasing, 13, 172
Ind, 13
index, of sequence element, 169
inductive, 13
R-inductive, 170
infimum, 172
initial algebra, 158
initial conversion, 101
initial labelling, 85
injection, 168
inner reduction, 132
innermost, 110
instance, 29
    most general common, 30
    of a conditional rule, 75
    of a reduction rule, 31
instantaneous description, 123
intermediate conversion, 102
interpretation, 141
intersection, multiset, 177
isomorphic, 158
Σ-isomorphism, 158
iteration, 168

join CTRS, 74
joinable, 74

König's Lemma, 176
Knuth–Bendix completion, 54

L-R theory, 163
labelled rules, 84
labelled term, 83
labelling
    initial, 85
landscape, 15
layer, 130
leaves, 175
left-linear, 36, 78
left-normal, 115
leftmost-fair, 117
length, 23
    of finite sequence, 169
    of reduction sequence, 8
length-non-increasing, 36
lexicographic combination, 148
lifting, 84, 85
linear term, 23
linear, (strict) order, 169
lists, 169
logicality, 39

many-sorted, 70
map(ping), 167
match, 29
maximum, 172
membership, of multiset, 176
mgci, 30
mgu, 30
minimum, 172

mixed term, 129
model, 40, 41
    initial, 159
modular, 128, 129
monochrome, 129
monomial, 145
monotone, 172
monotone algebra, 141
Morse sequence, 57
most general unifier, 30
multi-step, 98
multiplicity, of element of multiset, 176
multiset, 176

nested, 45, 48
NF, 13
node
    father, mother, 175
    of a tree, 174
    son, daughter, 175
non-ambiguous, 78, 80
non-collapsing, 36
non-duplicating, 36
non-erasing, 36, 108
non-overlapping, 78
normal CTRS, 74
normal form, 1, 12, 34
normal form property, 13
normalization
    complete, 111
    strong, 12
    weak, 12
    weakly innermost, 111
normalizing, 112
numeral, 55

occurrence, 24
one-hole context, 24
one-sorted, 70
one-step reduction, 32, 33
open corner, 101
order, 169
    (strict) linear, 169
    (strict) partial, 169
    partial, 169
    quasi-, 169
    strict (partial), 169
    total, 169
    well-, 169
    well-founded, 169
orthogonal, 78
    weakly, 79
orthogonal steps, 85
orthogonality, 78
outer reduction, 132
outermost-fair, 117
overlap, 44–46, 48, 80
    trivial, 46

paradoxical combinator, 63
parallel move, 89
Parallel Moves Lemma, 89, 90
Parallel Moves Proposition, 90, 98
parallel or, 81
parallel reduction, 89
parallel step, 89
partial order, 169
path
    in tree, 174
    maximal, 175
    reduction, 8
pattern, 45
PCP, 125
peak, 15
permutation equivalence, 106
perpetual, 118
PML, 90
PMP, 90, 98
polynomial, 145
    termination, 145
    interpretation, 139, 144
position, 26
Post's correspondence problem, 125
precedence, 140
predecessor, 81, 175
prefix, 24
preorder, 169
principal subterm, 130
Prism Theorem, 99
process algebra, 148
product, lexicographic, 173
product, of orders, 172
projection, 106
projection equivalence, 106
proper subterm, 24
pseudo-TRS, 33

quasi-order, 169
quotient algebra, 155
quotient, modulo a relation, 169

R-L theory, 163
rank, 130
recursive function, 57
recursive path order, 140
recursive program scheme, 69, 80
redex, 31, 32

created, 86, 88
inner, 132
outer, 132
redex pattern, 45
reduct, 8, 34
α, 8
one-step, 8
reduction, 8, 34
δ-, 81
fair, 115
Gross–Knuth, 113
innermost, 110
Kleene, 113
leftmost-fair, 117
multi-step, 98
outermost-fair, 117
parallel, 89
simultaneous, 98
underlinable, 94
underlined, 91
reduction diagram, 102
completed, 101, 102
reduction graph, 9
reduction order, 140
reduction path, 8
reduction relation, 1, 7
reduction rule, 31, *see also* rewrite rule
reduction sequence, 8, 34
indexed, 8
maximal, 13
reduction step, 32, *see also* rewrite step
atomic, 31
erasing, 108
reduction strategy, 112, *see also* strategy
refinement, 28
reflexive closure, 171
reflexive–transitive closure, 171
Reidemeister move, 3
relation, 167
acyclic, 171
antisymmetric, 169
binary, 167
descendant/ancestor, 85
equivalence, 169
inverse, 168
irreflexive, 169
reduction, 1
reflexive, 169
single-valued, 167
symmetric, 169
transitive, 169
well-founded, 169
renaming, 28
residual, 83, 86, 87
rewrite relation, 7
rewrite rule, 31, *see also* rule
conditional, 74
rewrite step, 32, *see also* step
critical, 108
rewriting
multi-step, 98
simultaneous, 98
right-linear, 36
root, 23
of tree, 174
RPS, 80, *see also* recursive program scheme
rule, *see also* reduction rule
collapsing, 36, 136
duplicating, 36, 136
erasing, 36
labelled, 84
left-linear, 36
right-linear, 36

Scott's Theorem, 127
secured, 115
self-application, 57
self-embedding, 121
semantical method, 139, 141
semantics, 40
semi-equational, 74
semi-equational CTRS, 74
semi-group, 120
semi-Thue system, 70
sequence, 168
ascending, 169
conversion, 9
descending, 169
empty, 169
finite, 169
reduction, 8
sub-, 168
sieve of Eratosthenes, 80
signature, 22
simultaneous rewriting, 98
size, of multiset, 177
SKIM, 64, 80
slice, 24
slope, 15
SN, 12, 34
soundness, 41, 156
special subterm, 130
splitting, 100
standard model, 41
step, *see also* reduction step

destructive, 131
empty, 101, 106
proper, 101
reduction, 8
S-term, 62
strategy
cofinal, 112
full substitution, 113
leftmost-innermost, 113
leftmost-outermost, 113
many-step, 112
normalizing, 112
one-step, 112
parallel-innermost, 113
parallel-outermost, 113
perpetual, 118
stream, 55
strict (partial) order, 169
string rewriting system, 70
strong confluence, 12
strong normalization
undecidable, 121
strongly normalizing, 12, 34
subcommutative, 10
subcontext, 24
subsequence, 169
substitution, 27
substitution instance, 29
subsume, 29
subsumption, 29
subterm, 24
principal, 130
proper, 24
special, 130
subtree, 175
successor, 175
supremum, 172
surjection, 168
symmetric closure, 171
syntactic accident, 34, 84
syntactical method, 140
Syracuse problem, 2

tautology checking, 4
term, 23
S-, 62
closed, 23
ground, 23
labelled, 83
linear, 23
underlined, 91
term algebra, 154
term algebra, ground, 154
term evaluation, 142
term model, 155
term model, ground, 155
term rewriting system, 21, 22, 33
applicative, 57, 66
complete, 42
first-order, 22
higher-order, 22
term tree, 25
terminating, 12
termination, 2
Thue–Morse sequence, 57
tiling, 101
TP, 10
transformational method, 140
transitive closure, 171
tree, 174
finitely branching, 175
term, 25
well-founded, 175
triangle property, 10, 98
trivial context, 24
trivial overlap, 46
TRS, 22, 33
underlined, 91
Turing machine, 122

UN, 13
underlinable, 94
underlined, 91
underlining, 91
unifier, 30
uniform halting problem, 120
uniform word problem, 157
union, multiset, 177
unique normal form property, 13

valid, 40
validity problem, 157
valley, 15
variable, 22
variant, 29

WCR, 10, 11
weak orthogonality, 79
weakly Church–Rosser, 10
weakly confluent, 10
weakly normalizing, 12, 34
weakly orthogonal, 79
well-founded, 140
well-founded monotone algebra, 141
well-order, 169
WF, 13
WIN, 110

WN, 12, 34, 68
word problem, 42, 70, 120, 158
uniform, 42