

---

# Term Rewriting Systems

---

Terese

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE  
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS  
The Edinburgh Building, Cambridge CB2 2RU, UK  
40 West 20th Street, New York, NY 10011-4211, USA  
477 Williamstown Road, Port Melbourne, VIC 3207, Australia  
Ruiz de Alarcón 13, 28014 Madrid, Spain  
Dock House, The Waterfront, Cape Town 8001, South Africa

<http://www.cambridge.org>

© Cambridge University Press 2003

This book is in copyright. Subject to statutory exception  
and to the provisions of relevant collective licensing agreements,  
no reproduction of any part may take place without  
the written permission of Cambridge University Press.

First published 2003

Printed in the United Kingdom at the University Press, Cambridge

Typeset by the author in Computer Modern using L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>

*A catalogue record for this book is available from the British Library*

*Library of Congress Cataloguing in Publication data*

ISBN 0 521 39115 6 hardback

---

# Contents

---

<b>Preface</b>	<b>xv</b>
<b>0 Introduction</b>	<b>1</b>
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
<b>1 Abstract reduction systems</b>	<b>7</b>
<i>Marc Bezem, Jan Willem Klop</i>	
1.1 Basic notions of ARSs . . . . .	7
1.2 Interrelations between ARS properties . . . . .	14
1.3 Specific ARS properties . . . . .	20
1.3.1 Confluence . . . . .	20
1.3.2 Normalization . . . . .	22
1.3.3 Miscellaneous . . . . .	22
<b>2 First-order term rewriting systems</b>	<b>24</b>
<i>Jan Willem Klop, Roel de Vrijer</i>	
2.1 Terms . . . . .	25
2.1.1 Contexts, occurrences . . . . .	26
2.1.2 Term trees . . . . .	28
2.1.3 Substitution . . . . .	31
2.1.4 Matching, subsumption, unification . . . . .	32
2.2 Definition of a TRS . . . . .	34
2.3 Reduction . . . . .	38
2.4 Equational systems . . . . .	41
2.5 Semantics . . . . .	43
2.6 Complete term rewriting systems and the word problem . . . . .	45
2.7 Overlap and critical pairs . . . . .	47
2.7.1 Overlapping redex patterns . . . . .	48
2.7.2 Critical pairs and the Critical Pair Lemma . . . . .	52
2.7.3 Roads to confluence . . . . .	57
2.8 Properties of critical pairs as criteria for confluence . . . . .	58
<b>3 Examples of TRSs and special rewriting formats</b>	<b>60</b>
<i>Jan Willem Klop, Roel de Vrijer</i>	
3.1 Numerals and streams . . . . .	60
3.2 Herbrand–Gödel computability . . . . .	62
3.3 Combinatory logic . . . . .	64

3.3.1	The combinators . . . . .	66
3.3.2	Miscellaneous facts about combinators . . . . .	69
3.3.3	Fixed points . . . . .	70
3.3.4	Mixed notation . . . . .	72
3.3.5	Currying . . . . .	73
3.4	Special rewriting formats . . . . .	75
3.4.1	Ground TRSs . . . . .	75
3.4.2	Constructor term rewriting systems . . . . .	76
3.4.3	Many-sorted term rewriting systems . . . . .	77
3.4.4	String rewriting systems . . . . .	78
3.5	Conditional term rewriting systems . . . . .	80
3.5.1	Conditional specifications . . . . .	80
3.5.2	Conditional rewriting . . . . .	81
3.6	The $\lambda\sigma$ -calculus . . . . .	85
<b>4</b>	<b>Orthogonality</b> . . . . .	<b>88</b>
	<i>Jan Willem Klop, Vincent van Oostrom, Roel de Vrijer</i>	
4.1	Introduction . . . . .	88
4.2	Descendants and residuals . . . . .	92
4.3	Confluence via the parallel moves lemma . . . . .	99
4.4	Underlinable reductions . . . . .	101
4.5	Complete developments . . . . .	106
4.6	Completion of reduction diagrams . . . . .	110
4.6.1	Elementary diagrams . . . . .	110
4.6.2	Reduction diagrams . . . . .	112
4.6.3	Permutation equivalence . . . . .	116
4.7	Inductive confluence proofs . . . . .	118
4.7.1	Confluence via parallel rewriting . . . . .	120
4.7.2	Inductively defined multi-steps . . . . .	124
4.8	Non-erasing reductions . . . . .	125
4.9	Reduction strategies . . . . .	129
4.9.1	Relaxing the constraints in $\mathbb{F}_{lm}$ , $\mathbb{F}_{GK}$ and $\mathbb{F}_{po}$ . . . . .	133
4.9.2	Standard reductions in orthogonal left-normal TRSs . . . . .	135
4.10	Confluence for weakly orthogonal TRSs . . . . .	137
4.11	Confluence for weakly orthogonal CTRSs . . . . .	141
4.11.1	Stable conditions . . . . .	142
4.11.2	Weakly orthogonal normal CTRSs . . . . .	144
4.11.3	Conditional linearization . . . . .	147
<b>5</b>	<b>Properties of rewriting: decidability and modularity</b> . . . . .	<b>149</b>
	<i>Jan Willem Klop, Roel de Vrijer</i>	
5.1	Decidability . . . . .	149
5.2	Easy (un)decidability results . . . . .	150
5.3	Undecidability of strong normalization . . . . .	151
5.3.1	Undecidability of SN via Turing machines . . . . .	152
5.3.2	Undecidability via register machines . . . . .	156
5.3.3	Reducing Post's correspondence problem to SN . . . . .	156

5.4	Undecidability properties of combinatory logic . . . . .	158
5.5	Undecidability of normal forms in conditional term rewriting systems	159
5.6	Decidability of WN for recursive program schemes . . . . .	161
5.7	Modularity . . . . .	163
5.7.1	Basic notions concerning disjoint sums of TRSs . . . . .	163
5.7.2	Modularity of weak normalization . . . . .	167
5.8	Modularity of confluence . . . . .	168
5.9	The case of strong normalization . . . . .	174
5.10	Other modularity results . . . . .	178
<b>6</b>	<b>Termination</b> . . . . .	<b>181</b>
	<i>Hans Zantema</i>	
6.1	Introduction . . . . .	181
6.2	Semantical methods . . . . .	183
6.2.1	Well-founded monotone algebras . . . . .	183
6.2.2	Polynomial interpretations . . . . .	187
6.2.3	Polynomial interpretations modulo AC . . . . .	194
6.2.4	Lexicographic combinations . . . . .	196
6.2.5	Other examples . . . . .	198
6.3	A hierarchy of termination . . . . .	199
6.3.1	Simple termination . . . . .	199
6.3.2	Total termination . . . . .	205
6.3.3	The hierarchy . . . . .	207
6.4	Syntactical methods . . . . .	210
6.4.1	Recursive path order . . . . .	211
6.4.2	Justification of recursive path order . . . . .	216
6.4.3	Extensions of recursive path order . . . . .	223
6.4.4	Knuth–Bendix order . . . . .	225
6.5	Transformational methods . . . . .	227
6.5.1	Basic transformations . . . . .	228
6.5.2	Dummy elimination . . . . .	231
6.5.3	Applying abstract commutation . . . . .	234
6.5.4	Semantic labelling . . . . .	241
6.5.5	The dependency pair method . . . . .	250
6.5.6	Type introduction . . . . .	256
<b>7</b>	<b>Completion of equational specifications</b> . . . . .	<b>260</b>
	<i>Inge Bethke</i>	
7.1	Equational specifications: syntax and semantics . . . . .	260
7.2	Initial algebra semantics . . . . .	266
7.3	Conditional equational deduction . . . . .	271
7.4	Critical pair completion . . . . .	273
7.5	An abstract formulation of completion . . . . .	282
7.6	Inductive proofs by consistency . . . . .	288
7.7	<i>E</i> -unification . . . . .	290

<b>8</b>	<b>Equivalence of reductions</b>	<b>301</b>
	<i>Vincent van Oostrom, Roel de Vrijer</i>	
8.1	Introduction . . . . .	301
8.1.1	Orthogonality of steps . . . . .	304
8.1.2	Equivalence of reductions . . . . .	307
8.1.3	Causal equivalence . . . . .	308
8.1.4	Permutation equivalence . . . . .	310
8.1.5	Labelling equivalence . . . . .	311
8.1.6	Standardization equivalence . . . . .	313
8.1.7	Trace equivalence . . . . .	314
8.1.8	Projection equivalence . . . . .	315
8.2	Rewriting systems revisited . . . . .	316
8.2.1	Abstract rewriting systems revisited . . . . .	317
8.2.2	Term rewriting systems revisited . . . . .	321
8.3	Permutation . . . . .	329
8.3.1	Structural equivalence . . . . .	330
8.3.2	Permutation equivalence . . . . .	335
8.4	Labelling . . . . .	338
8.4.1	Labelling abstract rewriting systems . . . . .	339
8.4.2	Unwinding abstract rewriting systems . . . . .	344
8.4.3	Labelling term rewriting systems . . . . .	348
8.4.4	Hyland–Wadsworth labelling . . . . .	353
8.4.5	Boudol–Khasidashvili labelling . . . . .	355
8.4.6	Lévy labelling . . . . .	356
8.5	Standardization . . . . .	360
8.5.1	Parallel standard reductions . . . . .	363
8.5.2	Selection parallel standardization . . . . .	367
8.5.3	Inversion parallel standardization . . . . .	371
8.5.4	Standard reductions . . . . .	377
8.5.5	Selection standardization . . . . .	380
8.5.6	Inversion standardization . . . . .	381
8.5.7	Left-normal term rewriting systems . . . . .	383
8.6	Tracing . . . . .	386
8.6.1	Tracing positions . . . . .	388
8.6.2	Tracing slices . . . . .	395
8.6.3	Unwinding term rewriting systems . . . . .	400
8.6.4	Label–tracing . . . . .	405
8.6.5	Reconstruction for Lévy unwinding . . . . .	415
8.6.6	Unwinding TRSs as ARSs . . . . .	421
8.6.7	Semi-standardization . . . . .	421
8.6.8	Reconstruction for Boudol–Khasidashvili unwinding . . . . .	426
8.7	Projection . . . . .	428
8.7.1	Abstract residual systems . . . . .	430
8.7.2	Abstract projection equivalence . . . . .	438
8.7.3	The reflexive–transitive closure of a residual system . . . . .	442
8.7.4	Term residual systems . . . . .	449

8.7.5	Term projection equivalence . . . . .	452
8.8	Applications to orthogonal term rewriting systems . . . . .	454
8.8.1	The Parallel Proofs Lemma . . . . .	454
8.8.2	Removing redundancy in labels . . . . .	458
8.8.3	Standardization in left-normal orthogonal systems . . . . .	459
8.8.4	Weakly orthogonal term rewriting systems . . . . .	463
8.9	Braids . . . . .	467
8.9.1	Equivalence of braids . . . . .	468
8.9.2	Orthogonality of braids . . . . .	470
<b>9</b>	<b>Strategies</b> . . . . .	<b>475</b>
	<i>Vincent van Oostrom, Roel de Vrijer</i>	
9.1	Strategies revisited . . . . .	478
9.1.1	Abstract rewriting strategies revisited . . . . .	478
9.1.2	Term rewriting strategies revisited . . . . .	482
9.2	Neededness . . . . .	485
9.2.1	Normalization of the needed strategy . . . . .	487
9.2.2	Hyper-normalization of the needed strategy . . . . .	489
9.2.3	Existence and undecidability of needed redexes . . . . .	496
9.2.4	Strongly sequential term rewriting systems . . . . .	499
9.2.5	Head neededness . . . . .	503
9.3	Fairness . . . . .	505
9.3.1	Fair strategies . . . . .	506
9.3.2	Outermost-fair strategies . . . . .	509
9.3.3	Applications to positional strategies . . . . .	512
9.4	Optimality . . . . .	515
9.4.1	The internal needed strategy . . . . .	516
9.4.2	The needed family strategy . . . . .	518
9.4.3	Sharing graph rewriting . . . . .	521
9.4.4	Optimal sharing . . . . .	528
9.5	Pessimality . . . . .	530
9.5.1	The limit strategy . . . . .	531
9.5.2	The zoom-in strategy . . . . .	534
9.6	Computable one-step strategies . . . . .	535
9.6.1	A recursive normalizing strategy . . . . .	537
9.6.2	A recursive cofinal strategy . . . . .	541
9.6.3	Simulating multi-step strategies . . . . .	546
<b>10</b>	<b>Lambda calculus</b> . . . . .	<b>548</b>
	<i>Inge Bethke</i>	
10.1	Basics . . . . .	548
10.2	Fundamental theorems . . . . .	554
10.2.1	Confluence of $\lambda\beta$ - and $\lambda\beta\eta$ -calculus . . . . .	554
10.2.2	Finiteness of developments . . . . .	557
10.2.3	Parallel moves . . . . .	562
10.2.4	Standardization and normalization . . . . .	564
10.3	Lambda definability . . . . .	571

10.4	Extensions of $\lambda$ -calculus . . . . .	572
10.5	Simply typed $\lambda$ -calculus . . . . .	581
<b>11</b>	<b>Higher-order rewriting</b>	<b>588</b>
	<i>Femke van Raamsdonk</i>	
11.1	Introduction . . . . .	588
11.2	Higher-order rewriting systems . . . . .	589
11.2.1	Terms . . . . .	590
11.2.2	Rewriting . . . . .	595
11.2.3	Related formats . . . . .	604
11.3	Combinatory reduction systems . . . . .	608
11.3.1	Terms . . . . .	608
11.3.2	Rewriting . . . . .	610
11.3.3	Related formats . . . . .	614
11.4	Comparing CRSs and HRSs . . . . .	617
11.4.1	From CRSs to HRSs . . . . .	618
11.4.2	From HRSs to CRSs . . . . .	623
11.5	Termination . . . . .	627
11.5.1	Finite developments . . . . .	628
11.5.2	Termination models . . . . .	633
11.5.3	Recursive path order . . . . .	641
11.6	Confluence . . . . .	642
11.6.1	Orthogonality . . . . .	642
11.6.2	Confluence by orthogonality . . . . .	647
11.6.3	Confluence by weak orthogonality . . . . .	656
11.6.4	Development closed . . . . .	661
11.6.5	Critical pairs . . . . .	665
<b>12</b>	<b>Infinitary rewriting</b>	<b>668</b>
	<i>Richard Kennaway, Fer-Jan de Vries</i>	
12.1	Introduction . . . . .	668
12.2	Infinitary term rewriting systems . . . . .	670
12.3	Strongly converging reductions . . . . .	672
12.4	Infinitary lambda calculus . . . . .	676
12.5	Descendants and developments . . . . .	679
12.6	Tiling diagrams . . . . .	684
12.7	The Compression Lemma . . . . .	689
12.8	A partial Church–Rosser property . . . . .	691
12.9	Meaningless terms . . . . .	697
12.9.1	Concepts of meaninglessness . . . . .	697
12.9.2	Consequences of meaninglessness . . . . .	699
12.9.3	Proofs . . . . .	701
12.9.4	Examples of classes of meaningless terms . . . . .	708
12.10A	refinement of infinitary lambda calculus . . . . .	709



<b>13 Term graph rewriting</b>	<b>712</b>
<i>Erik Barendsen</i>	
13.1 Introduction . . . . .	712
13.2 Graphs . . . . .	714
13.3 Equational presentations of graphs . . . . .	722
13.3.1 Equational characterization of bisimilarity . . . . .	724
13.3.2 Syntactic characterizations of homomorphisms . . . . .	726
13.3.3 Flattening and hiding . . . . .	728
13.4 Rewriting . . . . .	730
13.5 Adequacy of graph rewriting for term rewriting . . . . .	735
13.6 Terms with recursion . . . . .	738
<b>14 Advanced ARS theory</b>	<b>744</b>
<i>Marc Bezem, Jan Willem Klop, Vincent van Oostrom</i>	
14.1 Reduction diagrams . . . . .	744
14.2 Confluence by decreasing diagrams . . . . .	751
14.3 Reduction modulo equivalence . . . . .	767
<b>15 Rewriting-based languages and systems</b>	<b>776</b>
<i>Jan Heering, Paul Klint</i>	
15.1 Introduction . . . . .	776
15.2 Application areas . . . . .	777
15.3 Systems . . . . .	779
15.4 Implementation of term rewriting . . . . .	788
<b>A Mathematical background</b>	<b>790</b>
<i>Marc Bezem</i>	
A.1 Relations, functions and orders . . . . .	790
A.2 Ordinals . . . . .	799
A.3 Inductive definitions . . . . .	801
A.4 Metric topology . . . . .	807
A.5 Infinite combinatorics . . . . .	810
A.6 Multisets . . . . .	820
<b>References</b>	<b>826</b>
<b>List of notations</b>	<b>857</b>
<b>Index of authors</b>	<b>864</b>
<b>Index of subjects</b>	<b>869</b>

---

# 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:

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

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  $\rightarrow$  rather than  $=$ , even though the expressions are equal in the sense that they all denote the same number (24). The relation  $\rightarrow$  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

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

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  $\square$ , indicating the place where the elementary step is to take place. In the first example above, the elementary step  $3 + 5 \rightarrow 8$  is performed in the context  $\square \cdot (1 + 2)$ . This means that  $\square$  is  $3 + 5$  before, and 8 after, the elementary step, yielding the simplification  $(3 + 5) \cdot (1 + 2) \rightarrow 8 \cdot (1 + 2)$ . (Brackets are not part of the term, but are auxiliary symbols preventing false readings of the term.) The next elementary step is  $1 + 2 \rightarrow 3$ , performed in the context  $8 \cdot \square$ , yielding the simplification  $8 \cdot (1 + 2) \rightarrow 8 \cdot 3$ . Finally, the elementary step  $8 \cdot 3 \rightarrow 24$  is performed in the context  $\square$ , 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:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow \dots$$

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

$$1 \rightarrow 2 \rightarrow \dots \rightarrow 11 \rightarrow 12 \rightarrow 1 \rightarrow \dots$$

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

$$n \rightarrow n'$$

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

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

Note that we have both decrease (in the case  $n/2$ ) and increase (in the case  $3n + 1$ ). It is an open question<sup>1</sup> 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.

---

<sup>1</sup>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  $\mathbf{S} : \mathbb{N} \rightarrow \mathbb{N}$ . 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:

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

As an example we exhibit the following reduction sequence.

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

Normal forms are 0, 1, 2,  $\dots$ , but also  $\text{minus}(0, 1)$  and  $\text{quot}(2, 0)$ . 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

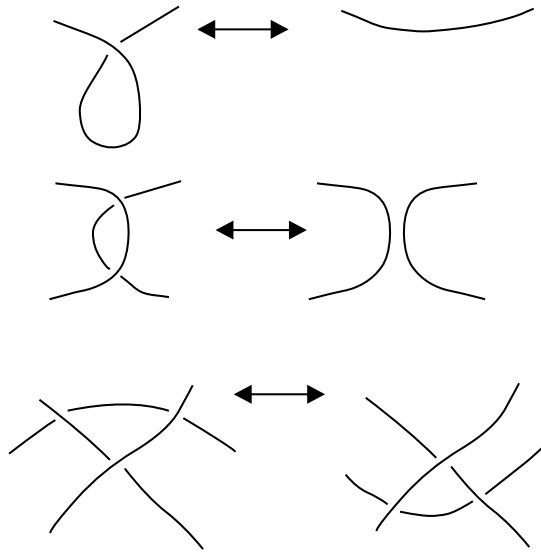


Figure 1: Reidemeister moves

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-

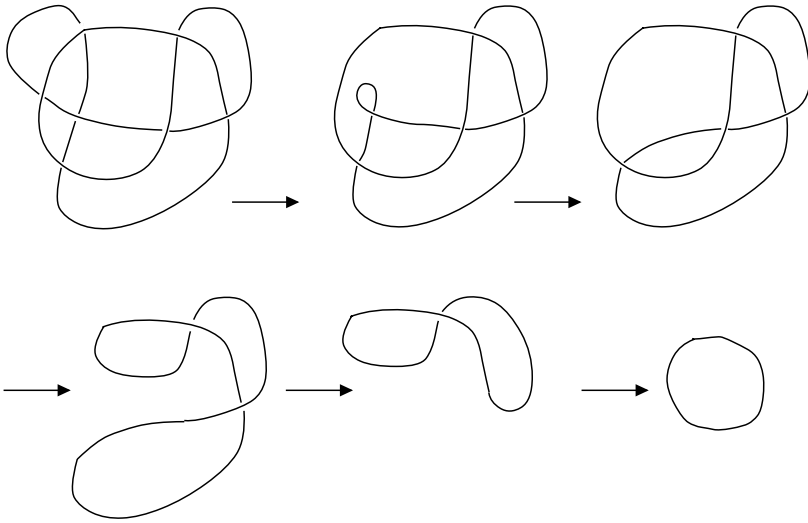


Figure 2: Un-knotting

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 Section 14.3.)

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.

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

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  $\cdot$ .

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

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

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.