Handbook of Logic in Computer Science

Volume 2
Background: Computational Structures

Edited by
S. ABRAMSKY, DOV M. GABBAY,
and T. S. E. MAIBAUM

OXFORD SCIENCE PUBLICATIONS

ISBN 0 19 8537611

Handbook of Logic in Computer Science

Volume 2

Background: Computational Structures

Edited by

S. ABRAMSKY
Professor of Computing Science

DOV M. GABBAY
Professor of Computing Science

and

T. S. E. MAIBAUM
Professor of Foundations of Software Engineering

Imperial College of Science, Technology and Medicine
London

Volume Co-ordinator

DOV M. GABBAY

CLARENDON PRESS · OXFORD
1992

Oxford University Press, Walton Street, Oxford OX2 6DP
Oxford New York Toronto
Delhi Bombay Calcutta Madras Karachi
Petaling Jaya Singapore Hong Kong Tokyo
Nairobi Dar es Salaam Cape Town
Melbourne Auckland
and associated companies in
Berlin Ibadan

Oxford is a trade mark of Oxford University Press

Published in the United States
by Oxford University Press, New York

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

Library of Congress Cataloging in Publication Data
Handbook of logic in computer science / edited by S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum.
v. 2
Contents: — v. 2. Background, computational structures.
1. Computer science. 2. Logic, Symbolic and mathematical.
I. Abramsky, S. II. Gabbay, Dov M., 1945– . III. Maibaum, Thomas S. E., 1947– . QA76.H2785 1992 92–510

ISBN 0–19–853761–1

Typeset by the authors
using TEX
Printed and bound in
Great Britain by Biddles Ltd.,
Guildford and Kings Lynn

HANDBOOK OF LOGIC IN COMPUTER SCIENCE

Editors

S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum

HANDBOOKS OF LOGIC IN COMPUTER SCIENCE
and
ARTIFICIAL INTELLIGENCE AND LOGIC PROGRAMMING

Handbook of Logic in Computer Science

Volume 1 Background: Mathematical structures
Volume 2 Background: Computational structures
Volume 3 Semantic structures
Volume 4 Semantic modelling
Volume 5 Theoretical methods in specification and verification
Volume 6 Logical methods in computer science

Handbook of Logic in Artificial Intelligence and Logic Programming

Volume 1 Logical foundations
Volume 2 Deduction methodologies
Volume 3 Nonmonotonic reasoning and uncertain reasoning
Volume 4 Epistemic and temporal reasoning
Volume 5 Logic programming

Preface

We are happy to present the first volumes of the *Handbook of Logic in Computer Science*. Logic is now widely recognized to be one of the foundational disciplines of computing and has found applications in virtually all aspects of the subject, from software engineering and hardware to programming language and artificial intelligence. There is a growing need for an in-depth survey of the application of logic in computer science and AI. The *Handbook of Logic in Computer Science* and its companion, the *Handbook of Logic in Artificial Intelligence and Logic Programming* have been created in response to this need.

We see the creation of the Handbook as a combination of authoritative exposition, comprehensive survey, and fundamental research exploring the underlying unifying themes in the various areas. The intended audience is graduate students and researchers in the areas of computing and logic, as well as other people interested in the subject. We assume as background some mathematical sophistication. Much of the material will also be of interest to logicians and mathematicians.

The tables of contents of the volumes were finalized after extensive discussions between handbook authors and second readers. The first two volumes present the background logic and mathematics extensively used in computer science. The point of view is application oriented. The other four volumes present major areas in which the methods are used. These include Volume 3 — Semantic Structures; Volume 4 — Semantic Modelling; Volume 5 — Specification and Verification; and Volume 6 — Logical Methods.

The chapters, which in many cases are of monographic length and scope, are written with emphasis on possible unifying themes. The chapters have an overview, introduction, and a main body. A final part is dedicated to more specialized topics.

Chapters are written by internationally renowned researchers in the respective areas. The chapters are co-ordinated and their contents were discussed in joint meetings.

Each chapter has been written using the following procedures:

1. A very detailed table of contents was discussed and co-ordinated at several meetings between authors and editors of related chapters. The discussion was in the form of a series of lectures by the authors to everyone present. Once an agreement was reached on the detailed table of contents the authors wrote a draft and sent it to the editors and to other related authors. For each chapter there is a second reader (the first reader is the author) whose job it has been to scrutinize the

chapter together with the editors. The second reader's role is very important and has required effort and serious involvement with the authors.
Second readers for this volume are:
Chapter 1: Term Rewriting Systems — J.-J. Levy
Chapter 2: Lambda Calculus — R. Hindley
Chapter 3: Algorithmic Proof Systems — R. Milner
Chapter 4: Designing a Theorem Prover — R. Milner
Chapter 5: Modal and Temporal Logics — A. Pneuli and R. Turner.

2. Once this process was completed (i.e. drafts seen and read by a large enough group of authors), there were other meetings on several chapters in which authors lectured on their chapters and faced the criticism of the editors and audience. The final drafts were prepared after these meetings.

3. We attached great importance to group effort and co-ordination in the writing of chapters. The first two parts of each chapter, namely the Introduction–Overview and Main Body, are not completely under the discretion of the author, as he/she had to face the general criticism of all the other authors. Only the third part of the chapter is entirely for the authors' own tastes and preferences.

The Handbook meetings were generously financed by OUP, by SERC contract SO/809/86, by the Department of Computing at Imperial College, and by several anonymous private donations.

We would like to thank our colleagues, authors, second readers, and students for their effort and professionalism in producing the manuscripts for the Handbook. We would particularly like to thank the staff of OUP for their continued and enthusiastic support, and Mrs Jane Spurr, our OUP Administrator for her dedication and efficiency.

London The Editors
May 1992

Contents

Elements of algorithmic proof

Designing a theorem prover

Modal and temporal logics

Term Rewriting Systems

J. W. Klop[1]

Contents

Abstract

Term rewriting systems (TRSs) play an important role in various areas, such as abstract data type specifications, implementations of functional programming languages and automated deduction. In this chapter we introduce several of the basic concepts and facts for TRSs. Specifically, we discuss abstract reduction systems; general TRSs including an account of Knuth–Bendix completion and (E-)unification; orthogonal TRSs and reduction strategies; strongly sequential orthogonal TRSs. The paper concludes with a discussion of conditional term rewriting systems. The em-

[1]Research partially supported by ESPRIT project 432: Meteor (until Sept. 1989) and ESPRIT BRA projects 3020: Integration and 3074: Semagraph (since July 1989).

phasis throughout the chapter is on providing information of a syntactic nature.

1 Introduction

The concept of a term rewriting system (TRS) is paradigmatic for the study of computational procedures. Already half a century ago, the λ-calculus, probably the most well-known TRS, played a crucial role in mathematical logic with respect to formalizing the notion of computability; much later the same TRS figured in the fundamental work of Scott, Plotkin and others, leading to a break-through in the denotational semantics of programming languages. More recently, the related system of combinatory logic was shown to be a very fruitful tool for the implementation of functional languages. Even more recently another related family of TRSs, that of categorical combinatory logic, has emerged, yielding a remarkable connection between concepts from category theory and elementary steps in machine computations.

Term rewriting systems are attractive because of their simple syntax and semantics—at least those TRSs that do not involve bound variables such as λ-calculus, but involve the rewriting of terms from a first-order language. This simplicity facilitates a satisfactory mathematical analysis. On the other hand they provide a natural medium for implementing computations, and in principle even for parallel computations. This feature makes TRSs interesting for the design of parallel reduction machines.

Another field where TRSs play a fundamental role concerns the analysis and implementation of abstract data type specifications (consistency properties, computability theory, decidability of word problems, theorem proving).

The aim of the present paper is to give an introduction to several key concepts in the theory of term rewriting, providing where possible some of the details. At various places some 'exercises' are included. These contain additional information for which proofs are relatively easy; they are not primarily meant to have an educational purpose, if only because the distribution of the exercises is not very uniform.

The present introduction starts at a level of 'rewriting' which is as abstract as possible and proceeds by considering term rewriting systems which have ever more 'structure'. Thus we start with abstract reduction systems, which are no more than sets equipped with some binary ('rewrite') relations. A number of basic properties and facts can already be stated on this level.

Subsequently, the abstract reductions are specialized to reductions (rewritings) of terms. For such general TRSs a key issue is to prove the

termination property; we present one of the major and most powerful termination proof methods, recursive path orderings, in a new formulation designed to facilitate human understanding (rather than practical implementation). Proving termination is of great importance in the area of Knuth–Bendix completions. Here one is concerned, given an equational specification, to construct a TRS which is both confluent and terminating and which proves the same equations as the original specification. If the construction is successful, it yields a positive solution to the validity problem of the original equational specification. (Nowadays there are also several other applications of Knuth–Bendix-like completion methods, such as 'inductionless induction' and 'computing with equations'. For a survey of such applications we refer to [Dershowitz and Jouannaud, 1990].)

Also in Section 2, we explain the basic ideas of Knuth–Bendix completion together with an interesting recent 'abstract' approach of [Bachmair *et al.*, 1986] to prove the correctness of Knuth–Bendix completion algorithms. We also present an elegant unification algorithm, and likewise for 'E-unification'.

In Section 3 we impose more 'structure' on TRSs, in the form of an 'orthogonality' requirement (non-ambiguity and left-linearity). For such orthogonal TRSs a sizeable amount of theory has been developed, both syntactically and semantically. Here we will almost exclusively be concerned with the syntactical aspects; for semantical aspects we refer to [Boudol, 1985], [Berry and Lévy, 1979] , [Guessarian, 1981]. Basic theorems (confluence, the parallel moves lemma, Church's theorem, O'Donnell's theorem) are presented, where possible with some proof sketch. Also in this section we survey the most important facts concerning reduction strategies for orthogonal TRSs, strategies aiming at finding normal forms whenever possible. Section 3 concludes with an explanation of the beautiful theory of [Huet and Lévy, 1979] of (strongly) sequential TRSs. Such TRSs possess a 'good' reduction strategy.

In the final section (4) we consider TRSs with conditional rewrite rules.

Some important topics have not found their way into this introduction. Most notable are: rewriting modulo a set of equations, proof-by-consistency procedures, and graph rewriting. For information about the first two we refer to [Bachmair, 1988] and [Dershowitz and Jouannaud, 1990] (1990), for graph rewriting one may consult [Barendregt *et al.*, 1987].

This chapter is an extension of the short survey/tutorial [Klop, 1987]; also most of the material in [Klop, 1985] is included here.

2 Abstract reduction systems

Many of the basic definitions for and properties of term rewriting systems (TRSs) can be stated more abstractly, viz. for sets equipped with one or more binary relations. As it is instructive to see which definitions and properties depend on the term structure and which are more basic, we start with a section about abstract reduction systems. Moreover, the concepts and properties of abstract reduction systems also apply to other rewrite systems than TRSs, such as string rewrite systems (Thue systems), tree rewrite systems, graph grammars. First we present a sequence of simple definitions.

Definition 2.0.1.

1. An *abstract reduction system* (ARS) is a structure $\mathcal{A} = \langle A, (\rightarrow_\alpha)_{\alpha \in I} \rangle$ consisting of a set A and a sequence of binary relations $\rightarrow_\alpha$ on A, also called reduction or rewrite relations. Sometimes we will refer to $\rightarrow_\alpha$ as α. In the case of just one reduction relation, we also use $\rightarrow$ without more. (An ARS with just one reduction relation is called a 'replacement system' in [Staples, 1975], and a 'transformation system' in [Jantzen, 1988].) If for $a, b \in A$ we have $(a, b) \in \rightarrow_\alpha$, we write $a \rightarrow_\alpha b$ and call b a one-step (α-)*reduct* of a.

2. The transitive reflexive closure of $\rightarrow_\alpha$ is written as $\twoheadrightarrow_\alpha$. (More customary is the notation $\rightarrow_\alpha^*$, but we prefer the double arrow notation as we find it more convenient in diagrams.) So $a \twoheadrightarrow_\alpha b$ if there is a possibly empty, finite sequence of 'reduction steps' $a \equiv a_0 \rightarrow_\alpha a_1 \rightarrow_\alpha \cdots \rightarrow_\alpha a_n \equiv b$. Here $\equiv$ denotes identity of elements of A. The element b is called an (α-)reduct of a. The equivalence relation generated by $\rightarrow_\alpha$ is $=_\alpha$, also called the *convertibility* relation generated by $\rightarrow_\alpha$. The reflexive closure of $\rightarrow_\alpha$ is $\rightarrow_\alpha^=$. The transitive closure of $\rightarrow_\alpha$ is $\rightarrow_\alpha^+$. The converse relation of $\rightarrow_\alpha$ is $\leftarrow_\alpha$ or $\rightarrow_\alpha^{-1}$. The union $\rightarrow_\alpha \cup \rightarrow_\beta$ is denoted by $\rightarrow_{\alpha\beta}$. The composition $\rightarrow_\alpha \circ \rightarrow_\beta$ is defined by: $a \rightarrow_\alpha \circ \rightarrow_\beta b$ if $a \rightarrow_\alpha c \rightarrow_\beta b$ for some $c \in A$.

3. If α, β are reduction relations on A, we say that α *commutes weakly* with β if the diagram of Figure 2.1a holds, i.e. if $\forall a, b, c \in A \; \exists d \in A \, (c \leftarrow_\beta a \rightarrow_\alpha b \Rightarrow c \twoheadrightarrow_\alpha d \twoheadleftarrow_\beta b)$, or in a shorter notation: $\leftarrow_\beta \circ \rightarrow_\alpha \subseteq \twoheadrightarrow_\alpha \circ \twoheadleftarrow_\beta$. Further, α *commutes* with β if $\twoheadrightarrow_\alpha$ and $\twoheadrightarrow_\beta$ commute weakly. (This terminology differs from that of [Bachmair and Dershowitz, 1986], where α commutes with β if $\alpha^{-1} \circ \beta \subseteq \beta^{-1} \circ \alpha$.)

4. The reduction relation $\rightarrow$ is called *weakly confluent* or *weakly Church–Rosser* (WCR) if it is weakly self-commuting (see Figure 2.1b), i.e. if $\forall a, b, c \in A \; \exists d \in A \, (c \leftarrow a \rightarrow b \Rightarrow c \twoheadrightarrow d \twoheadleftarrow b)$. (The property WCR is also often called 'local confluence', e.g. in [Jantzen, 1988].)

5. $\to$ is *subcommutative* (notation $\mathrm{WCR}^{\leq 1}$) if the diagram in Figure 2.1c holds, i.e. if $\forall a, b, c \in A\ \exists d \in A\ (c \leftarrow a \to b \Rightarrow c \to^{=} d \leftarrow^{=} b)$.
6. $\to$ is *confluent* or is *Church–Rosser*, has the Church-Rosser property (CR) if it is self-commuting (see Figure 2.1d), i.e. $\forall a, b, c \in A\ \exists d \in A\ (c \twoheadleftarrow a \twoheadrightarrow b \Rightarrow c \twoheadrightarrow d \twoheadleftarrow b)$. Sometimes (6) is called 'confluent' and the situation as in Proposition 2.0.2(6) 'Church–Rosser'.

Proposition 2.0.2. *The following are equivalent:*

1. $\to$ *is confluent*
2. $\twoheadrightarrow$ *is weakly confluent*
3. $\twoheadrightarrow$ *is self-commuting*
4. $\twoheadrightarrow$ *is subcommutative*
5. *the diagram in Figure 2.1e holds, i.e.*

$$\forall a, b, c \in A\ \exists d \in A\ (c \leftarrow a \twoheadrightarrow b \Rightarrow c \twoheadrightarrow d \twoheadleftarrow b)$$

6. $\forall a, b \in A\ \exists c \in A\ (a = b \Rightarrow a \twoheadrightarrow c \twoheadleftarrow b)$ *(Here '=' is the convertibility relation generated by* $\to$*. See diagram in Figure 2.1f.)*

Definition 2.0.3. Let $\mathcal{A} = \langle A, \to \rangle$ be an ARS.

1. We say that $a \in A$ is a *normal form* if there is no $b \in A$ such that $a \to b$. Further, $b \in A$ *has a normal form* if $b \twoheadrightarrow a$ for some normal form $a \in A$.
2. The reduction relation $\to$ is *weakly normalizing* (WN) if every $a \in A$ has a normal form. In this case we also say that $\mathcal{A}$ is WN.
3. $\mathcal{A}$ (or $\to$) is *strongly normalizing* (SN) if every reduction sequence $a_0 \to a_1 \to \cdots$ eventually must terminate. (Other terminology: $\to$ is *terminating*, or *noetherian*.) If the converse reduction relation $\leftarrow$ is SN, we say that $\mathcal{A}$ (or $\to$) is SN^{-1}.
4. $\mathcal{A}$ (or $\to$) has the *unique normal form property* (UN) if $\forall a, b \in A (a = b$ & a, b are normal forms $\Rightarrow a \equiv b)$.
5. $\mathcal{A}$ (or $\to$) has the *normal form property* (NF) if $\forall a, b \in A (a$ is normal form & $a = b \Rightarrow b \twoheadrightarrow a)$.
6. $\mathcal{A}$ (or $\to$) is *inductive* (Ind) if for every reduction sequence (possibly infinite) $a_0 \to a_1 \to \cdots$ there is an $a \in A$ such that $a_n \twoheadrightarrow a$ for all n.
7. $\mathcal{A}$ (or $\to$) is *increasing* (Inc) if there is a map $|\ |\colon A \to \mathbb{N}$ such that $\forall a, b \in A\ (a \to b \Rightarrow |\ a\ | < |\ b\ |)$. Here $\mathbb{N}$ is the set of natural numbers with the usual ordering $<$.
8. $\mathcal{A}$ (or $\to$) is *finitely branching* (FB) if for all $a \in A$ the set of one-step reducts of a, $\{b \in A \mid a \to b\}$, is finite. If the converse reduction