

Lecture Notes in Computer Science

922

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Advisory Board: W. Brauer D. Gries J. Stoer

Heiko Dörr

Efficient Graph Rewriting and Its Implementation



Springer

Series Editors

Gerhard Goos

Universität Karlsruhe

Vincenz-Priessnitz-Straße 3, D-76128 Karlsruhe, Germany

Juris Hartmanis

Department of Computer Science, Cornell University

4130 Upson Hall, Ithaca, NY 14853, USA

Jan van Leeuwen

Department of Computer Science, Utrecht University

Padualaan 14, 3584 CH Utrecht, The Netherlands

Author

Heiko Dörr

Institut für Informatik, Freie Universität Berlin

Takustrasse 9, D-14195 Berlin, Germany

Library of Congress Cataloging-in-Publication Data. Dörr, Heiko, 1962– Efficient graph rewriting and its implementation/Heiko Dörr. p.cm. – (Lecture notes in computer science; 922) Includes bibliographical references and index. ISBN 3-540-60055-8 (alk. paper). – ISBN 0-387-60055-8 (U.S.: alk. paper). 1. Rewriting systems (Computer science) I. Title. II. Series. QA267. D67 1995

005. 13' 1–dc20

95-30326 CIP

CR Subject Classification (1991): F.4.2, D.3.2, D.3.4, D.2.1

ISBN 3-60055-8 Springer-Verlag Berlin Heidelberg New York

CIP data applied for

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1995

Printed in Germany

Typesetting: Camera-ready by author

SPIN: 10485935

06/3142-543210 - Printed on acid-free paper

Preface

Graph rewriting systems have come of age. In autumn 1994, the 25th anniversary of the first publication in this area was celebrated at the 5th Workshop on Graph Grammars and their Applications to Computer Science. In the interim, the subject has evolved. The current situation can be described by a three-stage model. At the very low level there is a common idea of graph rewriting as the basic mechanism, where a graph is transformed by the application of a rewriting rule.

In the second stage, this mechanism is expressed in several ways. Usually, two so-called approaches are distinguished: the algorithmic (or set-theoretic) and the algebraic approach. Both provide a formalisation of graph rewriting. They give a precise semantics to the idea of graph transformation and, hence, allow for a formal treatment. In that sense, they are similar to the semantics of programming languages.

The upper stage is partitioned into several branches. At one extreme, theoretical studies on the generational power, on semantic constructs, or on restricted formalisms are undertaken. At the other extreme, specifications of real-world systems, or implementations of rewriting environments are developed. Because the individual problems to solve are complicated enough, the branches are not very aware of each other.

The monograph builds bridges between various areas of interest: 1) it studies a class of graph rewriting systems which is very suitable for an efficient execution; 2) it presents a compilation approach to the implementation of an environment for graph rewriting; 3) it develops an implementation of a functional programming language to show that and how the presented ideas apply to real-world problems.

This publication is my dissertation which I submitted to the *Fachbereich Mathematik und Informatik* at the *Freie Universität Berlin*. It was finished in December 1994, and presents the results of the research carried out at the *Institut für Informatik*. In that respect I would like to thank several people who accompanied my work. Prof. Elfriede Fehr provided an excellent environment for my studies and gave me any support I requested; Prof. Gregor Engels introduced me to graph rewriting and encouraged my progress with valuable comments; the GraGra-AG, in particular Prof. Hartmut Ehrig and Gabi Taenzer, provided a refuge from the graph rewriting Diaspora; Prof. Raúl Rojas posed the very question initiating the whole theoretical consideration; Albert Zündorf and Andreas Schürr provided me with the newest versions of PROGRES and the additional information which was not included in the documentation; my colleagues at work provided an important social background; Elke Kasimir implemented most parts of the environment and was a very valuable critic;

Peter Hofmann developed a compiler for functional languages based on the ideas presented in Chapter 6; Gaye Rochow and John Kelly helped me by proof-reading the manuscript to eliminate the mistakes made by a non-native speaker: Thanks to you all.

Berlin, April 1995

Heiko Dörr

Contents

1 Introduction

1.1 Graph Rewriting Everywhere? — The Purpose of Graph Rewriting Systems	1
1.2 The Major Flaw of Graph Rewriting - Its Complexity	4
1.3 The Plan of Attack - Finding the Gap	5
1.4 Outline	7

2 Graph Rewriting Systems — The Basic Concepts

2.1 Vertices, Edges, and Labels make a Graph — Preliminary Definitions	10
2.2 Algorithmic Graph Rewriting Systems — The Basic Formalism	13
2.3 An Abstract Machine for Labelled Subgraph Matching	24
2.4 Summary and Related Work	31

3 UBS-Graph Rewriting Systems — Matching Subgraphs in Constant Time

3.1 Unique Labels — Singularities in Derived Graphs	38
3.1.1 Unique Vertex Labels	38
3.1.2 Unique Edge Labels	43
3.2 Label Triples — Detecting Dead Rules and Saving Memory	47
3.2.1 Definition and Properties of Label Triples	47
3.2.2 Approximation of the Set of Label Triples	49
3.3 Strong V-Structures — The Branching Points	56
3.3.1 Strong V-Structures and Matching in Constant Time	58
3.3.2 Determination of a Bypassing Connected Enumeration	63
3.3.3 Conditions for the Introduction of Strong V-Structures	65
3.3.4 Approximation of the Set of Strong V-Structures	81
3.4 Summary and Related Work	85

4 Programmed Attributed Graph Rewrite Systems — An Advanced Modelling Formalism

4.1	Attributed Graph Rewriting Systems — Extension I	92
4.1.1	The Formalism	93
4.1.2	Application of the Analyses	99
4.2	Programmed Graph Rewriting Systems — Extension II	101
4.2.1	The Syntax	105
4.2.2	The Failure Semantics	106
4.2.3	The Collection Semantics	110
4.2.4	Analysis of Further Control Structures	114
4.3	Summary and Related Works	117

5 The Abstract Machine for Graph Rewriting — Supporting a Fast Implementation

5.1	Optimization of the Application Test for Rule Sets	125
5.2	The Graph Rewriting Environment	136
5.2.1	The State of the Core Abstract Machine	138
5.2.2	Matching Instructions	141
5.2.3	Structural Graph Updates	145
5.2.4	Evaluation of Embedding Descriptions	146
5.2.5	Attribute Evaluation	147
5.2.6	Control Instructions	149
5.3	Code Generation for the Abstract Machine	149
5.3.1	The Syntax	150
5.3.2	The Code Generation Rules	152
5.4	Improvements by Rule Set Optimization	157
5.5	Summary and Related Work	160

6 A Graphical Implementation of Functional Languages — A Case Study in UBS-Graph Rewriting Systems

6.1	Aspects of Functional Programming	165
6.2	Translating Functions to Trees — Graph Reduction approaches Graph Rewriting	168
6.2.1	A Graphical Notation for Functional Expressions	168
6.2.2	Function Definitions as Rewriting Rules	170
6.2.3	Translating Expressions	171
6.2.4	Translating Function Definitions	176

6.3	Sharing — Graph Reduction Meets Graph Rewriting	181
6.4	Normal Order Reduction — Enforcing a Deterministic Evaluation Order	185
6.4.1	A Stack of Spine Pointers	187
6.4.2	Deconstruction of the Spine Stack	191
6.4.3	Unwinding the Spine	192
6.5	The Printing Mechanism	197
6.6	The Translation of Lazy Evaluation is UBS	201
6.6.1	A Mandatory Set of Prohibited Strong V-Structures	202
6.6.2	None of the Prohibited Strong V-Structures Occur	206
6.7	Summary and Related Work	213
7	Conclusions	217
Appendix A	List of Figures and Tables	223
Appendix B	Implementation of a Functional Program	225
Appendix C	References	255
Appendix D	Index	261