Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

348

- P. Deransart B. Lorho
- J. Małuszyński (Eds.)

Programming Languages Implementation and Logic Programming

International Workshop PLILP '88 Orléans, France, May 16–18, 1988 Proceedings



Springer-Verlag Berlin Heidelberg New York London Paris Tokyo

Editorial Board

D. Barstow W. Brauer P. Brinch Hansen D. Gries D. Luckham C. Moler A. Pnueli G. Seegmüller J. Stoer N. Wirth

Editors

Pierre Deransart INRIA-Rocquencourt, Domaine de Voluceau B.P. 105, F-78153 Le Chesnay Cedex, France

Bernard Lorho Université d'Orléans, Faculté des Sciences Laboratoire d'Informatique Fondamentale (LIFO) B.P. 6759, F-45067 Orléans Cedex 2, France

Jan Maluszyński Department of Computer and Information Science Linköping University S-58183 Linköping, Sweden



CR Subject Classification (1987): F.4.1-2, D.3.1, D.3.4, F.3.3, I.2.3

ISBN 3-540-50820-1 Springer-Verlag Berlin Heidelberg New York ISBN 0-387-50820-1 Springer-Verlag New York Berlin Heidelberg

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 other ways, and storage in data banks. Duplication of this publication or parts thereof is only permitted under the provisions of the German Copyright Law of September 9, 1965, in its version of June 24, 1985, and a copyright fee must always be paid. Violations fall under the prosecution act of the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1989

Printed in Germany

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr. 2145/3140-543210 – Printed on acid-free paper

PREFACE

PLILP '88, the first international Workshop on Programming Languages Implementation and Logic Programming, was held from May 16 to May 18, 1988 in Orléans. PLILP '88 has been organized by the Laboratoire d'Informatique Fondamentale d'Orléans (LIFO-Université d'Orléans) and Institut National d'Informatique et d'Automatique (INRIA-Rocquencourt).

The aim of the workshop was to discuss whether research on the implementation of programming languages and research on logic programming can mutually benefit from each other's results. The intention was to bring together researchers from both fields, especially those working in the area of their intersection.

Problems such as formal specification of compilers and syntax-based editors, program analysis and program optimization have been traditionally studied by implementors of algorithmic languages and have resulted in a number of well-established notions, formalisms and techniques. At the same time, an increasing number of people use logic programming as a way of specifying compilers or other programming environment tools, taking advantage of the relatively high level of logic programming and the growing efficiency of Prolog implementations.

On the other hand, research on logic programming raises the questions of analysis of logic programs and their optimization. These are motivated primarily by compiler construction for logic programs, by studies on the methodology of logic programming and by the attempts to amalgamate logic programming and functional programming.

Research in the field of logic programming, including its applications to the implementation of other programming languages, may or may not refer to the well-known results of the other field. In the first case the field of logic programming may benefit from these results. For example application of LR parsing techniques may contribute to a more efficient implementation of definite clause grammars. On the other hand, techniques of logic programming may contribute to the development of the other field. As an example, one may consider the use of logic programs for compiler specification.

The purpose of the workshop was to review the techniques developed in one (or both) of the fields which could also be of some help in the other one and to facilitate the transfer of expertise. It seems important to compare notions used in both fields : pointing out similarities between them may prevent rediscovering results already known, while studying the differences may contribute to the transfer of technology.

The workshop consisted of a series of invited talks and a panel discussion. This book presents some of the most significant talks.

We gratefully acknowledge the financial support provided by the following institutions : - INRIA.

- Université d'Orléans,

- GRECO de Programmation et Outils pour l'Intelligence Artificielle du CNRS.

Le Chesnay, Orléans, Linköping December 1988 Pierre Deransart Bernard Lorho Jan Maluszynski

Table of Contents



ļ

Functional Programming and Logic Programming
Static Analysis of Functional Programs with Logical Variables Gary Lindström
Towards a Clean Amalgamation of Logic Programs with External Procedures Staffan Bonnier and Jan Maluszynski
Abstract Interpretation in Logic Programming
An Application of Abstract Interpretation in Source Level Program Transformation Daniel De Schreye and Maurice Bruynooghe
A Tool to Check the Non-Floundering Logic Programs and Goals Roberto Barbuti and Maurizio Martelli
Towards a Framework for the Abstract Interpretation of Logic Programs Ulf Nilsson 68
Logic Programming in Compiler Writing
An Implementation of Retargetable Code Generators in Prolog Annie Despland, Monique Mazaud and Raymond Rakotozafy
Towards a "Middle Road" Methodology for Writing Code Generators Feliks Kluzniak and Miroslawa Milkowska
A Compiler Written in Prolog : the Véda Experience Jean-François Monin
Grammars
Coupled Context-Free Grammar as a Programming Paradigm Yoshiyuki Yamashita and Ikuo Nakata
A Bottom-Up Adaptation of Earley's Parsing Algorithm Frédéric Voisin
Using an Attribute Grammar as a Logic Program Günter Riedewald and Uwe Lämmel
Attribute Grammars and Logic Programming
Structure Sharing in Attribute Grammars Henning Christiansen
A Semantic Evaluator Generating System in Prolog Pedro Rangel Henriques
A Grammatical View of Logic Programming Pierre Deransart and Jan Maluszynski

Attribute	Grammars	in	Logic	Programming
-----------	----------	----	-------	-------------

Compiling Typol with Attribute Grammars Isabelle Attali	252
Formal Specification of a Prolog Compiler Michael Hanus	273

Logic Programming for Programming Environments

Formal Specification of Interactive Languages Using Definite Clause Grammars Weidong Dang	283
Using Logic Databases in Software Development Environments Patrizia Asirelli and Paola Inverardi	292

Static Analysis Of Functional Programs With Logical Variables*

Gary Lindstrom Department of Computer Science University of Utah Salt Lake City, Utah 84112 USA

Abstract

It has recently been discovered by several researchers that logical variables, even under unconditional unification, can significantly increase the expressive power of functional programming languages. Capabilities added under this extension include (i) support for use before binding of variables, e.g. in constructing functionally attributed parse trees; (ii) computation by constraint intersection, e.g. polymorphic type checking; (iii) "micro object" support for object oriented programming, e.g. for direct message delivery, and (iv) monotonic refinement of complex data structures, e.g. function evaluation by graph reduction.

In contrast to the fundamental producer-consumer orientation of pure functional programming, there is no single "producer" of the value of a logical variable. All efficient implementations of pure functional programming rely on direct access to value sources, and exploit the resulting uni-directional information flow (e.g. by dataflow, demand propagation, or normal order graph reduction). One may therefore ask whether these implementation techniques can be augmented to accommodate the "isotropic" information flow associated with logical variables. In a previous paper we showed that the answer is largely affirmative for fine grain (S, K, I) combinator reduction implementations. We now outline an approach that adapts this technique to larger granularity combinators through a static analysis technique that estimates both operator strictness (graph partitioning into co-evaluation equivalence classes) and mode effects on logical variables (e.g. "read-only" occurrences). Many advantages are achieved, but the impossibility of comprehensive flow analysis means that the resulting large grain combinators cannot comprise exclusively sequential code.

^{&#}x27;This research was supported in part by grant CCR-8704778 from the National Science Foundation, and by an unrestricted gift to the University of Utah from Telefonaktiebolaget LM Ericsson, Stockholm, Sweden.