Andrei Voronkov (Ed.)

Logic Programming and Automated Reasoning

4th International Conference, LPAR '93
St. Petersburg, Russia, July 13-20, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editor

Jörg Siekmann
University of Saarland
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken 11, FRG

Volume Editor

Andrei Voronkov
Department of Computer Science, University of Uppsala
Polacksbacken 1, Box 311, S-75105 Uppsala, Sweden

CR Subject Classification (1991): F.4.1, I.2.3, D.1.6

ISBN 3-540-56944-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-56944-8 Springer-Verlag New York Berlin Heidelberg

Printed in Germany

Typesetting: Camera ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

LPAR is an international conference traditionally held in Russia since 1990. LPAR'93 is organized by the Russian Association for Logic Programming. It aims at bringing together researchers interested in logic programming and automated reasoning. The research in logic programming grew out of the research in automated reasoning in the early 1970s. Later, the implementation techniques used in logic programming were used in implementing theorem proving systems. Results from both fields are used in deductive databases.

The scientific program of LPAR'93 includes 5 invited talks, 35 talks and 4 advanced tutorials. 35 papers included in this volume were selected from 84 submitted papers. In addition, Peter Wegner made it possible to prepare a written version of his invited talk. During the conference several systems implemented on IBM PC and compatibles were demonstrated.

There are many people involved in the organization of LPAR'93. I wish to personally thank Oleg Gusikhin, Irina Freidson, Igor' Litvinov, Vladimir Popov, Igor' Rents, Tania Rybina, Michail Simuni, Yuri Shcheglyuk.

I gratefully acknowledge financial sponsorship by the Commission of the European Communities.

Uppsala, May 1993 Andrei Voronkov

Program Committee

Dmitri Boulanger (Catholic University Leuven)
Mats Carlsson (SICS, Kista)
Philippe Codognet (INRIA Rocquencourt)
Danny De Schreye (Catholic University Leuven)
Norbert Eisinger (ECRC, Munich)
Harald Ganzinger (Max-Planck-Institut für Informatik, Saarbrücken)
Ryuzo Hasegawa (ICOT, Tokyo)
Steffen Hölldobler (Technische Hochschule Darmstadt)
Deepak Kapur (SUNY at Albany)
Jean-Louis Lassez (IBM Thomas J.Watson Research Center, Yorktown Heights)
Alexander Leitsch (Technische Universität Wien)
Giorgio Levi (University of Pisa)
John Lloyd (University of Bristol)
Ewing Lusk (Argonne National Laboratory)
Dale Miller (University of Pennsylvania)
Jack Minker (University of Maryland)
Gregory Mints (Stanford University)
Alan Mycroft (University of Cambridge)
Lee Naish (University of Melbourne)
Hans-Jürgen Ohlbach (Max-Planck-Institut für Informatik, Saarbrücken)
Michel Parigot (University of Paris 7)
Frank Pfenning (Carnegie Mellon University)
Vladimir Sazonov (Program Systems Institute, Pereslavl-Zalesski)
Marek Sergot (Imperial College)
Mark Stickel (SRI International, Menlo Park)
Pascal Van Hentenryck (Brown University)
Konstantin Vershinin (Institute for Cybernetics, Kiev)
Andrei Voronkov (Uppsala University) — chairman
Nail Zamov (Kazan' University)

Organizing Committee

Eugene Dantsin (Electrical Engineering Institute, St. Petersburg)
Robert Freidson (Electrical Engineering Institute, St. Petersburg) — chairman
Andrei Voronkov (ECRC, Munich)

Andreas Nonnengart (Max-Planck-Institute, Saarbrücken)
Frank O'Carroll (ICOT, Tokyo)
Doug Palmer (University of Melbourne)
Remo Pareschi (ECRC, Munich)
Lawrence Paulson (University of Cambridge)
Dino Pedreschi (University of Pisa)
Shekhar Pradhan (University of Maryland)
Sanjiva Prasad (ECRC, Munich)
Sophie Renault (INRIA - Roquencourt)
Eike Ritter (University of Cambridge)
Igor Romanenko (University of Kiev)
François Rouaix (INRIA - Roquencourt)
Paul Rozière (University of Paris 7)
Vladimir Rudenko (University of Kiev)
Carolina Ruiz (University of Maryland)
Dan Sahlin (SICS, Kista)
Gernot Salzer (Technische Universität Wien)
Torsten Schaub (Technische Hochschule Darmstadt)
Manfred Schmidt-Schauß (Universität Frankfurt)
Kees Schuerman (ECRC, Munich)
Yasuyuki Shinai (ICOT, Tokyo)
Marianne Simonot (University of Paris 7)
Gert Smolka (DFKI Saarbrücken)
Rolf Socher-Ambrosius (Max-Planck-Institute, Saarbrücken)
Harald Søndergaard (University of Melbourne)
Sury Sripada (ECRC, Munich)
B.Steffen (Max-Planck-Institute, Saarbrücken)
Yukihrde Takayama (ICOT, Tokyo)
Michael Thielsche (Technische Hochschule Darmstadt)
Andreas Tönne (Max-Planck-Institute, Saarbrücken)
André Véron (ECRC, Munich)
Benjamin Werner (University of Paris 7)
Mark Wallace (ECRC, Munich)
Richard Zach (Technische Universität Wien)

Invited Speakers

Alan Bundy (Edinburgh University)
Hervé Gallaire (Xerox France)
Ryuzo Hasegawa (ICOT)
Peter Wegner (Brown University)
Nail Zamov (Kazan' University)

Contents

Entailment and Disentailment of Order-Sorted Feature Constraints 1
Hassan Aït-Kaci, Andreas Podelski

Computing Extensions of Default Logic — Preliminary Report 19
Grigoris Antoniou, Elmar Langetepe, Volker Sperschneider

Prolog with Arrays and Bounded Quantifications 28
Jonas Barklund, Johan Bevemyr

Linear 0-1 Inequalities and Extended Clauses 40
Peter Barth

Search Space Pruning by Checking Dynamic Term Growth 52
Stefan Brüning

A Proof Search System for a Modal Substructural Logic
Based on Labelled Deductive Systems 64
Hiu Fai Chau

Consistency Checking of Automata Functional Specifications 76
Anatoli N. Chebotarev, Marina K. Morokhovets

Yet Another Application for Toupie: Verification of
Mutual Exclusion Algorithms 86
Marc-Michel Corsini, Alain Griffault, Antoine Rauzy

Parsing with DCG-Terms 98
Andrew Davison

A First Order Resolution Calculus with Symmetries 110
Uwe Egly

Ordered Paramodulation and Resolution as Decision Procedure 122
Christian G. Fermüller and Gernot Salzer

Static Analysis of Prolog with Cut 134
Gilberto Filé, Sabina Rossi

Entailment and Disentailment of Order-Sorted Feature Constraints

Hassan Aït-Kaci and Andreas Podelski

Digital Equipment Corporation, Paris Research Laboratory
85, avenue Victor Hugo, 92500 Rueil-Malmaison, France

{hak,podelski}@prl.dec.com

Abstract. LIFE uses matching on order-sorted feature structures for passing arguments to functions. As opposed to unification which amounts to normalizing a conjunction of constraints, solving a matching problem consists of deciding whether a constraint (guard) or its negation are entailed by the context. We give a complete and consistent set of rules for entailment and disentailment of order-sorted feature constraints. These rules are directly usable for relative simplification, a general proof-theoretic method for proving guards in concurrent constraint logic languages using guarded rules.

1 Introduction

LIFE [5] extends the computational paradigm of Logic Programming in two essential ways:

- using a data structure richer than that provided by first-order constructor terms; and,
- allowing interpretable functional expressions as *bona fide* terms.

The first extension is based on ψ-terms which are attributed partially-ordered sorts denoting sets of objects [1, 2]. In particular, ψ-terms generalize first-order constructor terms in their rôle as data structures in that they are endowed with a unification operation denoting type intersection.

The second extension deals with building into the unification operation a means to reduce functional expressions using definitions of interpretable symbols over data patterns. The basic insight is that unification is no longer seen as an atomic operation by the resolution rule. Indeed, since unification amounts to normalizing a conjunction of equations, and since this normalization process commutes with resolution, these equations may be left in a normal form that is not a fully solved form. In particular, if an equation involves a functional expression whose arguments are not sufficiently instantiated to match a *definiens* of the function in question, it is simply left untouched. Resolution may proceed until the arguments are *proven* to match a definition from the accumulated constraints in the context [3]. This simple idea turns out invaluable in practice.

This technique—delaying reduction and enforcing determinism by allowing only equivalence reductions—is called *residuation* [3]. It does not have to be limited to functions. Therefore, we explain it for the general case of relations. Intuitively, the arguments of a relation which are constrained by the guard are its input parameters and correspond to the arguments of a function. This has been used as an implicit control mechanism in general concurrent constraint logic programming schemes; *e.g.*, the logic of guarded Horn-clauses studied by Maher [11], Concurrent Constraint Programming (CCP) [12], and Kernel Andorra Prolog (KAP) [9]. These schemes are parameterized with respect to an abstract class of constraint systems. An incremental test for entailment and disentailment between constraints is needed for advanced control mechanisms such as delaying, coroutining, synchronization, committed choice, and deep constraint propagation. LIFE is formally an instance of this scheme, namely a

CLP language using a constraint system based on order-sorted feature (OSF) structures [5]. It employs a related, but limited, suspension strategy to enforce deterministic functional application. Roughly, these systems are concurrent thanks to a new effective discipline for procedure parameter-passing that can be described as "call-by-constraint-entailment" (as opposed to Prolog's call-by-unification).

The most direct way to explain the issue is with an example. In LIFE, one can define functions as usual; say:

$$fact(0) \quad \rightarrow 1.$$
$$fact(N : int) \rightarrow N * fact(N - 1).$$

More interesting is the possibility to compute with partial information. For example:

$$minus(negint) \rightarrow posint.$$
$$minus(posint) \rightarrow negint.$$
$$minus(zero) \quad \rightarrow zero.$$

Let us assume that the symbols *int*, *posint*, *negint*, and *zero* have been defined as sorts with the approximation ordering such that *posint*, *zero*, *negint* are pairwise incompatible subsorts of the sort *int* (*i.e.*, $posint \wedge zero = \bot, negint \wedge zero = \bot, posint \wedge negint = \bot$). This is declared in LIFE as $int := \{posint; zero; negint\}$. Furthermore, we assume the sort definition $posint := \{posodd; poseven\}$; *i.e.*, *posodd* and *poseven* are subsorts of *posint* and mutually incompatible.

The LIFE query $Y = minus(X : poseven)?$ will return $Y = negint$. The sort *poseven* of the actual parameter is incompatible with the sort *negint* of the formal parameter of the first rule defining the function *minus*. Therefore, that rule is skipped. The sort *poseven* is more specific than the sort *posint* of the formal parameter of the second rule. Hence, that rule is applicable and yields the result $Y = negint$.

The LIFE query $Y = minus(X : string)$ will fail. Indeed, the sort *string* is incompatible with the sort of the formal parameter of every rule defining *minus*.

Thus, in order to determine which of the rules, if any, defining the function in a given functional expression will be applied, two tests are necessary:

- verify whether the actual parameter is more specific than or equal to the formal parameter;
- verify whether the actual parameter is at all compatible with the formal parameter.

What happens if both of these tests fail? For example, consider the query consisting of the conjunction:

$$Y = minus(X : int), X = minus(zero)?$$

Like Prolog, LIFE follows a left-to-right resolution strategy and examines the equation $Y = minus(X : int)$ first. However, both foregoing tests fail and deciding which rule to use among those defining *minus* is inconclusive. Indeed, the sort *int* of the actual parameter in that call is neither more specific than, nor incompatible with, the sort *negint* of the first rule's formal parameter. Therefore, the function call will *residuate* on the variable X. This means that the functional evaluation is suspended pending more information on X. The second goal in the query is treated next. There, it is found that the actual parameter is incompatible with the first two rules and is the same as the last rule's. This allows reduction and binds X to *zero*. At this point, X has been instantiated and therefore the residual equation pending on X can be reexamined. Again, as before, a redex is found for the last rule and yields $Y = zero$.

The two tests above can in fact be worded in a more general setting. Viewing data structures as constraints, "more specific" is simply a particular case of constraint entailment.

We will say that a constraint *disentails* another whenever their conjunction is unsatisfiable; or, equivalently, whenever it entails its negation. In particular, first-order matching is deciding entailment between constraints consisting of equations over first-order terms. Similarly, deciding unifiability of first-order terms amounts to deciding "compatibility" in the sense used informally above.

The suspension/resumption mechanism illustrated in our example is repeated each time a residuated actual parameter becomes more instantiated from the context; *i.e.*, through solving other parts of the query. Therefore, it is most beneficial for a practical algorithm testing entailment and disentailment to be incremental. This means that, upon resumption, the test for the instantiated actual parameter builds upon partial results obtained by the previous test. One outcome of the results presented in this paper is that it is possible to build such a test; namely, an algorithm deciding simultaneously two problems in an incremental manner—entailment and disentailment. The technique that we have devised to do that is called *relative simplification* of constraints.

We have organized this paper as follows. In Section 2, we review background on our OSF formalism. This is for the sake of staying self-contained since its technical notation and terminology is pervasive in this paper's presentation. In Section 3, we give rules for incrementally deciding entailment and disentailment of OSF constraints, and we make explicit the effectuality of relative simplification. In Section 4, we prove the termination of the rules. In Section 5, we show the correctness and completeness of these rules. Section 6 establishes the property of independence of negated OSF constraints. Finally, we conclude in Section 7.

2 OSF Formalism

We introduce briefly the OSF formalism terminology and notation that we use. For a thorough investigation of these notions, the reader is referred to [5].

2.1 OSF algebras and OSF constraints

The building blocks of OSF algebras are sorts and features.

An *order-sorted feature signature* (or simply OSF signature) is a tuple $\langle \mathcal{S}, \leq, \wedge, \mathcal{F} \rangle$ such that:

- $\mathcal{S}$ is a set of *sorts* containing the sorts $\top$ and $\bot$;
- $\leq$ is a decidable partial order on $\mathcal{S}$ such that $\bot$ is the least and $\top$ is the greatest element;
- $\langle \mathcal{S}, \leq, \wedge \rangle$ is a lower semi-lattice ($s \wedge s'$ is called the greatest common subsort of sorts s and s');
- $\mathcal{F}$ is a set of *feature symbols*.

An OSF signature has the following interpretation. An *OSF algebra* over the signature $\langle \mathcal{S}, \leq, \wedge, \mathcal{F} \rangle$ is a structure:

$$\mathcal{A} = \langle\, D^{\mathcal{A}},\ (s^{\mathcal{A}})_{s \in \mathcal{S}},\ (\ell^{\mathcal{A}})_{\ell \in \mathcal{F}} \,\rangle$$

such that:

- $D^{\mathcal{A}}$ is a non-empty set, called the *domain* of $\mathcal{A}$ (or, universe);
- for each sort symbol s in $\mathcal{S}$, $s^{\mathcal{A}}$ is a subset of the domain; in particular, $\top^{\mathcal{A}} = D^{\mathcal{A}}$ and $\bot^{\mathcal{A}} = \emptyset$;
- the greatest lower bound (*GLB*) operation on the sorts is interpreted as the intersection; *i.e.*, $(s \wedge s')^{\mathcal{A}} = s^{\mathcal{A}} \cap s'^{\mathcal{A}}$ for two sorts s and s' in $\mathcal{S}$.
- for each feature ℓ in $\mathcal{F}$, $\ell^{\mathcal{A}}$ is a total unary function from the domain into the domain; *i.e.*, $\ell^{\mathcal{A}} : D^{\mathcal{A}} \mapsto D^{\mathcal{A}}$;