LNAI 2385

Jacques Calmet
Belaid Benhamou
Olga Caprotti
Laurent Henocque
Volker Sorge (Eds.)

Artificial Intelligence, Automated Reasoning, and Symbolic Computation

Joint International Conferences, AISC 2002 and Calculemus 2002
Marseille, France, July 2002
Proceedings

Springer

Jacques Calmet Belaid Benhamou
Olga Caprotti Laurent Henocque
Volker Sorge (Eds.)

Artificial Intelligence, Automated Reasoning, and Symbolic Computation

Joint International Conferences
AISC 2002 and Calculemus 2002
Marseille, France, July 1-5, 2002
Proceedings

Springer

Volume Editors

Jacques Calmet
University of Karlsruhe (TH)
Am Fasanengarten 5, Postfach 6980, D-76128 Karlsruhe, Germany
E-mail: calmet@ira.uka.de

Belaid Benhamou
Université de Provence, CMI
39 rue F. Juliot-Curie, 13453 Marseille Cedex 13, France
E-mail: Belaid.Benhamou@cmi.univ-mrs.fr

Olga Caprotti
Johannes Kepler University
Research Institute for Symbolic Computation (RISC-Linz)
A-4040 Linz, Austria
E-mail: ocaprott@risc.uni-linz.ac.at

Laurent Henocque
Université de la Méditerannée, ESIL
163 Avenue de Luminy, Marseille Cedex 09, France
E-mail: henocque@esil.univ-mrs.fr

Volker Sorge
University of Birmingham, School of Computer Science
Birmingham B15 2TT, United Kingdom
E-mail: V.Sorge@cs.bham.ac.uk

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Artificial intelligence, automated reasoning, and symbolic computation : joint international conferences ; proceedings / AISC 2002 and Calculemus 2002, Marseille, France, July 1 -5, 2002. Jacques Calmet ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2385 : Lecture notes in artificial intelligence)
ISBN 3-540-43865-3

CR Subject Classification (1998): I.2.1-4, I.1, G.1-2, F.4.1

ISSN 0302-9743
ISBN 3-540-43865-3 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2002
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN 10870512 06/3142 5 4 3 2 1 0

Preface

AISC 2002, the 6th international conference on Artificial Intelligence and Symbolic Computation, and Calculemus 2002, the 10th symposium on the Integration of Symbolic Computation and Mechanized Reasoning, were held jointly in Marseille, France on July 1-5, 2002. This event was organized by the three universities in Marseille together with the LSIS (Laboratoire des Sciences de l'Information et des Systèmes).

AISC 2002 was the latest in a series of specialized conferences founded by John Campbell and Jacques Calmet with the initial title "Artificial Intelligence and Symbolic Mathematical Computation" (AISMC) and later denoted "Artificial Intelligence and Symbolic Computation" (AISC). The scope is well defined by its successive titles.

AISMC-1 (1992), AISMC-2 (1994), AISMC-3 (1996), AISC'98, and AISC 2000 took place in Karlsruhe, Cambridge, Steyr, Plattsburgh (NY), and Madrid respectively. The proceedings were published by Springer-Verlag as LNCS 737, LNCS 958, LNCS 1138, LNAI 1476, and LNAI 1930 respectively.

Calculemus 2002 was the 10th symposium in a series which started with three meetings in 1996, two meetings in 1997, and then turned into a yearly event in 1998. Since then, it has become a tradition to hold the meeting jointly with an event in either symbolic computation or automated deduction.

Both events share common interests in looking at Symbolic Computation, each from a different point of view: Artificial Intelligence in the more general case of AISC and Automated Deduction in the more specific case of Calculemus. Holding the two conferences jointly should trigger interdisciplinary research, with the first results expected at AISC 2004 (Austria) and at Calculemus 2003.

This volume includes papers accepted for presentation at both AISC 2002 and Calculemus 2002. From the 52 contributions submitted, 17 full papers were accepted for AISC, plus 7 full papers and 2 system descriptions for the Calculemus program. In addition, the invited speakers' abstracts are included as part of this volume. Several work-in-progress contributions were accepted for presentation at Calculemus 2002 but are not published in these proceedings.

We would like to express our thanks to the members of the two program committees, to the AISC steering committee, to the Calculemus trustees, to the referees, and to the organizing committee. Finally, we gratefully thank the sponsors for their financial support. Names are listed on the following pages.

April 2002

Jacques Calmet
Belaid Benhamou
Olga Caprotti
Laurent Henocque
Volker Sorge

Organization

Both AISC 2002 and CALCULEMUS 2002 were organized by the three universities of Marseille: L'Université de Provence (Aix-Marseille I), L'Université de la Méditerranée (Aix-Marseille II), La Faculté des Sciences de Saint-Jerôme (Aix-Marseille III), and the LSIS (Laboratoire des Sciences de l'Information et des Systèmes).

Conference Organization (AISC)

Conference and local chair:	Belaid Benhamou (Univ. de Provence, Aix-Marseille I)
Program chair:	Laurent Henocque (Univ. de la Méditerranée, Aix-Marseille II)
Steering committee:	Jacques Calmet (Univ. Karlsruhe, Germany)
	John Campbell (Univ. College London, UK)
	Eugenio Roanes-Lozano (Univ. Complutense de Madrid, Spain)

Symposium Organization (CALCULEMUS)

Program chairs:	Olga Caprotti (RISC, Hagenberg, Austria)
	Volker Sorge (Univ. of Birmingham, UK)
Local chair:	Belaid Benhamou (Univ. de Provence, Aix-Marseille I)

Local Committee (AISC / CALCULEMUS)

Gilles Audemard	(Univ. de Provence, Aix-Marseille I)
Belaid Benhamou	(Univ. de Provence, Aix-Marseille I)
Philippe Jegou	(Fac. de Saint Jerôme, Aix-Marseille III)
Laurent Henocque	(Univ. de la Méditerranée, Aix-Marseille II)
Pierre Siegel	(Univ. de Provence, Aix-Marseille I)

Program Committee (AISC)

Luigia C. Aiello	(Univ. La Sapienza, Roma, Italy)
Jose A. Alonso	(Univ. de Sevilla, Spain)
Michael Beeson	(San Jose State Univ., USA)
Belaid Benhamou	(Univ. de Provence, France)
Greg Butler	(Univ. Concordia, Montreal, Canada)
Simon Colton	(Univ. of York, UK)
Jim Cunningham	(Imperial College London, UK)
James Davenport	(Univ. of Bath, UK)
Carl van Geem	(LAAS-CNRS, Toulouse, France)
Reiner Haehnle	(Univ. of Technology, Chalmers, Sweden)
Deepak Kapur	(Univ. New Mexico, USA)
Luis M. Laita	(Univ. Politecnica de Madrid, Spain)
Luis de Ledesma	(Univ. Politecnica de Madrid, Spain)
Eric Monfroy	(Univ. de Nantes, France)
Jose Mira	(UNED, Spain)
Ewa Orlowska	(Inst. Telecommunications, Warsaw, Poland)
Jochen Pfalzgraf	(Univ. Salzburg, Austria)
Jan Plaza	(Univ. Plattsburgh, USA)
Zbigniew W. Ras	(Univ. North Carolina, Charlotte, USA)
Tomas Recio	(Univ. de Santander, Spain)
Peder Thusgaard Ruhoff	(MDS, Proteomics, Denmark)
Pierre Siegel	(Univ. de Provence, France)
Andrzej Skowron	(Warsaw Univ., Poland)
John Slaney	(ANU, Canberra, Australia)
Viorica Sofronie-Stokkermans	(Max Planck Institut, Germany)
Karel Stokkermans	(Univ. Salzburg, Austria)
Carolyn Talcott	(Stanford Univ., USA)
Rich Thomason	(Univ. of Pittsburgh, USA)
Dongming Wang	(Univ. Paris VI, France)

Additional Referees (AISC)

W. Ahrendt	M. Ceberio	A. C. Norman
F. Benhamou	M. Damsbo	H. Wang
R. Bernhaupt	L. Bordeaux	
B. Buchberger	M. Jaeger	
O. Caprotti	P.A. Madsen	

Program Committee (CALCULEMUS)

Alessandro Armando	(Univ. of Genova, Italy)
Christoph Benzmüller	(Univ. of Saarbrücken, Germany)
Jacques Calmet	(Univ. of Karlsruhe, Germany)
Alessandro Coglio	(Kestrel Institute, Palo Alto, USA)
Arjeh Cohen	(Univ. of Eindhoven, The Netherlands)
Simon Colton	(Univ. of Edinburgh, Scotland, UK)
James Davenport	(Univ. of Bath, UK)
William M. Farmer	(McMaster Univ., Hamilton, Canada)
Thérèse Hardin	(Univ. de Paris VI, France)
Hoon Hong	(Univ. of North Carolina State, Raleigh, USA)
Manfred Kerber	(Univ. of Birmingham, UK)
Michael Kohlhase	(Carnegie Mellon Univ., Pittsburgh, USA)
Steve Linton	(Univ. of St. Andrews, Scotland, UK)
Ursula Martin	(Univ. of St. Andrews, Scotland, UK)
Julian Richardson	(Heriot-Watt Univ., Edinburgh, Scotland,UK)
Renaud Rioboo	(Univ. de Paris VI, France)
Roberto Sebastiani	(Univ. di Trento, Italy)
Andrew Solomon	(Univ. of Technology, Sydney, Australia)
Andrzej Trybulec	(Bialystok Univ., Poland)
Volker Weisspfenning	(Univ. of Passau, Germany)
Wolfgang Windsteiger	(RISC, Hagenberg, Austria)

Additional Referees (CALCULEMUS)

M. Benerecetti	M. Jaume	G. Norman
D. Doligez	V. Ménissier-Morain	S. Ranise
A. Fiedler	M. Moschner	

Sponsoring Institutions

L'Ecole d'Ingénieur en Informatique de Luminy, ESIL
L'Université de Provence, Aix-Marseille I
Le Conseil Général de Marseille
La Mairie de Marseille
Calculemus Project
CologNet, European Network of Excellence

Table of Contents

Constraint Acquisition*

Eugene C. Freuder

Cork Constraint Computation Centre
University College Cork, Cork, Ireland
e.freuder@4c.ucc.ie; www.4c.ucc.ie

Abstract. Many problems may be viewed as constraint satisfaction problems. Application domains range from construction scheduling to bioinformatics. Constraint satisfaction problems involve finding values for problem variables subject to restrictions on which combinations of values are allowed. For example, in scheduling professors to teach classes, we cannot schedule the same professor to teach two different classes at the same time. There are many powerful methods for solving constraint satisfaction problems (though in general, of course, they are NP-hard). However, before we can solve a problem, we must describe it, and we want to do so in an appropriate form for efficient processing. The Cork Constraint Computation Centre is applying artificial intelligence techniques to assist or automate this modelling process. In doing so, we address a classic dilemma, common to most any problem solving methodology. The problem domain experts may not be expert in the problem solving methodology and the experts in the problem solving methodology may not be domain experts.

* The author is supported by a Principal Investigator Award from Science Foundation Ireland.

Expressiveness and Complexity of Full First-Order Constraints in the Algebra of Trees

Alain Colmerauer

Laboratoire d'Informatique Fondamentale de Marseille, Université Aix-Marseille II, France, alain.colmerauer@lim.univ-mrs.fr

Extended Abstract

What can be expressed by constraints, in the algebra of trees, if quantifiers and all the logical connectors are allowed? What is the complexity of algorithms for solving such general first-order constraints? This talk answers these two questions.

Preliminaries. Let F be a set of function symbols. The algebra of trees consists of the set of trees, whose nodes are labelled by elements of F, together with the construction operations linked to the elements f of F. Such an operation, with f of arity n, is the mapping $(a_1, \ldots, a_n) \mapsto a$, where a is the tree, whose initial node is labelled f and whose sequence of daughters is $a_1, \ldots, a_n$. A general first-order constraint is a formula made from: variables, elements of F, the equality symbol $=$, the logical constants and connectors *true*, *false*, $\neg, \wedge, \vee$ and the usual quantifiers $\exists, \forall$.

Expressiveness. With respect to expressiveness, we show how to express a constraint of the form

$$\varphi^n(x,y) \overset{\text{def}}{=} \begin{bmatrix} \exists u_0 \ldots \exists u_n \\ x = u_0 \wedge \\ \varphi(u_0, u_1) \wedge \\ \varphi(u_1, u_2) \wedge \\ \cdots \\ \varphi(u_{n-1}, u_n) \wedge \\ u_n = y \end{bmatrix}$$

by an equivalent constraint $\psi_n(x,y)$, of size almost proportional to the size of the constraint $\varphi(x,y)$. More precisely, we show that there exists a constant c such that, for any n,

$$|\psi_n(x,y)^{\alpha(n)}| \leq c\,|\varphi(x,y)|,$$

where $\alpha(n)$ is the huge integer

$$\alpha(n) \overset{\text{def}}{=} \underbrace{2^{\cdots^{\left(2^{(2^2)}\right)}}}_{n}.$$

For $n = 5$, this integer is already larger than the number of atoms of the universe.

Complexity. With respect to complexity, by making use of the previous result, we show that there exists a constant d and an integer n_0 such that:

For any $n \geq n_0$, there exists a constraint ψ of size n, without free variables, such that any algorithm, which decides whether ψ is satisfied in the algebra of trees, executes at least $\alpha(\lfloor dn \rfloor)$ instructions.

Deduction versus Computation: The Case of Induction

Eric Deplagne and Claude Kirchner

LORIA & INRIA,
615 rue du Jardin Botanique, BP 101,
54602 Villers-lès-Nancy Cedex, Nancy, France.
{Eric.Deplagne,Claude.Kirchner}@loria.fr

Abstract. The fundamental difference and the essential complementarity between computation and deduction are central in computer algebra, automated deduction, proof assistants and in frameworks making them cooperating. In this work we show that the fundamental proof method of induction can be understood and implemented as either computation or deduction.
Inductive proofs can be built either explicitly by making use of an induction principle or implicitly by using the so-called induction by rewriting and inductionless induction methods. When mechanizing proof construction, explicit induction is used in proof assistants and implicit induction is used in rewrite based automated theorem provers. The two approaches are clearly complementary but up to now there was no framework able to encompass and to understand uniformly the two methods. In this work, we propose such an approach based on the general notion of deduction modulo. We extend slightly the original version of the deduction modulo framework and we provide modularity properties for it. We show how this applies to a uniform understanding of the so called induction by rewriting method and how this relates directly to the general use of an induction principle.

Summary

Induction is a fundamental proof method in mathematics. Since the emergence of computer science, it has been studied and used as one of the fundamental concepts to build mathematical proofs in a mechanized way. In the rising era of proved softwares and systems it plays a fundamental role in frameworks allowing to search for formal proofs. Therefore proofs by induction have a critical role in proof assistants and automated theorem provers. Of course these two complementary approaches of proof building use induction in very different ways. In proof assistants like COQ, ELF, HOL, Isabelle, Larch, NQTHM, PVS, induction is used explicitly since the induction axiom is applied in an explicit way: the human user or a clever tactics should find the right induction hypothesis as well as the right induction variables and patterns to conduct the induction steps. In automated theorem provers specific methods have been developed to automatically prove inductive properties. The most elaborated ones are based on term rewriting and saturation techniques. They are respectively called induction by rewriting and inductionless induction or proof by consistency. Systems that implement these ideas are Spike, RRL or INKA.

The latter automated methods have been studied since the end of the seventies and have shown their strengths on many practical examples from simple algebraic specifications to more complicated ones like the Gilbreath card trick. But what was intriguing from the conceptual point of view was the relationship between explicit and implicit induction: implicit induction was shown to prove inductive theorems, but the relationship with the explicit use of the induction principle was open.

In this work, we provide a framework to understand *both* approaches in a *unified* way. One important consequence is that it allows us to combine in a well-understood way automated and assisted proof search methods. This reconciliation of the two approaches will allow automated theorem provers and proof assistants to collaborate in a safe way. It will also allow proof assistants to embark powerful proof search tactics corresponding to implicit induction techniques. This corresponds to the deduction versus computation scheme advocated in [1] under the name of deduction modulo: we want some computations to be made blindly i.e. without the user interaction and in this case this corresponds to implicit induction; but one also needs to explicitly control deduction, just because we know this is unavoidable and this can also be more efficient.

It is thus not surprising to have our framework based on deduction modulo. This presentation of first-order logic relies on the sequent calculus modulo a congruence defined on terms and propositions. But since we need to formalize the induction axiom which is by essence a second-order proposition, we need to use the first-order representation of higher-order logic designed in [2]. In this formalism, switching from explicit induction to implicit one becomes clear and amounts to push into the congruence some of the inductive reasoning, then to apply standard automated reasoning methods to simplify the goal to be proved and possibly get a better representation of the congruence.

This work relies on the notions and notations of deduction modulo [1] as well as on the first-order presentation of higher-order logic presented in [2]. We refer to these two papers for full definitions, details and motivations of the framework. Using this new framework, we uniformly review the induction by rewriting method and show how it directly relates to the induction principle, thus providing proof theoretic instead of model theoretic proofs of this rewrite based method.

Consequently, since the proof method is completely proof theoretic, to any rewrite based inductive proof we can canonically associate an explicit proof in the sequent calculus, thus providing a proof assistant with all the necessary informations to replay the proof as needed.

References

1. Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. Rapport de Recherche 3400, Institut National de Recherche en Informatique et en Automatique, April 1998.
`ftp://ftp.inria.fr/INRIA/publication/RR/RR-3400.ps.gz`.
2. Gilles Dowek, Thérèse Hardin, and Claude Kirchner. HOL-$\lambda\sigma$ an intentional first-order expression of higher-order logic. *Mathematical Structures in Computer Science*, 11(1):21–45, 2001.

Integration of Quantifier Elimination with Constraint Logic Programming

Thomas Sturm

University of Passau, Germany
sturm@uni-passau.de
http://www.fmi.uni-passau.de/~sturm/

Abstract. We examine the potential of an extension of constraint logic programming, where the admissible constraints are arbitrary first-order formulas over some domain. Constraint solving is realized by effective quantifier elimination. The arithmetic is always exact. We describe the conceptual advantages of our approach and the capabilities of the current implementation CLP(RL). Supported domains are currently $\mathbb{R}$, $\mathbb{C}$, and $\mathbb{Q}_p$. For our discussion here we restrict to $\mathbb{R}$.

1 Constraint Logic Programming

Logic programming languages have emerged during the early seventies with Prolog by Colmerauer and Kowalski being the by far most prominent example. The major conceptual contribution was disconnecting *logic* from *control* [Kow79]. The programmer should not longer be concerned with specifying and coding algorithmic control structures but instead *declaratively* specify a problem within some formal logical framework (Horn clauses). The system would then provide a universal control algorithm (resolution) for solving the specified problem. Prolog became surprisingly successful during the eighties. This pure approach of declarative specification, however, turned out to be not sufficiently efficient. On the basis of the observation that the arithmetic capabilities of the processing machine had remained unused, there had then been numbers added to Prolog and *built-in predicates* on these numbers. This approach, however, was not compatible with the original idea of separating logic and control.

This dilemma has been resolved with the step from logic programming (LP) to *constraint logic programming* (CLP) around the mid of the eighties. CLP combines logic programming languages with *constraint solvers*. Constraint solving was another established declarative programming paradigm that had come into existence already in the early sixties in connection with graphics systems. A *constraint solving problem* is given by a finite set of *constraints*. A constraint is a relational dependence between several objects, variables, and certain functions on these numbers and variables. The type of objects and the admitted functions and relational dependences make up the *domain* of the constraint solver. One example are linear programming problems (the target function can be coded as an extra constraint). A *solution* of a constraint system is *one* binding of all variables such that all constraints are simultaneously satisfied. A constraint solver