LNCS 3018

Maurice Bruynooghe (Ed.)

Logic Based Program Synthesis and Transformation

13th International Symposium, LOPSTR 2003
Uppsala, Sweden, August 2003
Revised Selected Papers

Springer

Maurice Bruynooghe (Ed.)

Logic Based Program Synthesis and Transformation

13th International Symposium, LOPSTR 2003
Uppsala, Sweden, August 25-27, 2003
Revised Selected Papers

Springer

Volume Editor

Maurice Bruynooghe
Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, 3001 Heverlee, Belgium
E-mail: Maurice.Bruynooghe@cs.kuleuven.ac.be

Library of Congress Control Number: 2004107503

CR Subject Classification (1998): F.3.1, D.1.1, D.1.6, I.2.2, F.4.1

ISSN 0302-9743
ISBN 3-540-22174-3 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11010531 06/3142 5 4 3 2 1 0

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Lecture Notes in Computer Science

For information about Vols. 1–2987

please contact your bookseller or Springer-Verlag

Vol. 3093: S.K. Katsikas, S. Gritzalis, J. Lopez (Eds.), Public Key Infrastructure. XIII, 380 pages. 2004.

Vol. 3092: J. Eckstein, H. Baumeister (Eds.), Extreme Programming and Agile Processes in Software Engineering. XVI, 358 pages. 2004.

Vol. 3091: V. van Oostrom (Ed.), Rewriting Techniques and Applications. X, 313 pages. 2004.

Vol. 3089: M. Jakobsson, M. Yung, J. Zhou (Eds.), Applied Cryptography and Network Security. XIV, 510 pages. 2004.

Vol. 3085: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs. X, 409 pages. 2004.

Vol. 3084: A. Persson, J. Stirna (Eds.), Advanced Information Systems Engineering. XIV, 596 pages. 2004.

Vol. 3083: W. Emmerich, A.L. Wolf (Eds.), Component Deployment. X, 249 pages. 2004.

Vol. 3078: S. Cotin, D.N. Metaxas (Eds.), Medical Simulation. XVI, 296 pages. 2004.

Vol. 3077: F. Roli, J. Kittler, T. Windeatt (Eds.), Multiple Classifier Systems. XII, 386 pages. 2004.

Vol. 3076: D. Buell (Ed.), Algorithmic Number Theory. XI, 451 pages. 2004.

Vol. 3074: B. Kuijpers, P. Revesz (Eds.), Constraint Databases and Applications. XII, 181 pages. 2004.

Vol. 3073: H. Chen, R. Moore, D.D. Zeng, J. Leavitt (Eds.), Intelligence and Security Informatics. XV, 536 pages. 2004.

Vol. 3070: L. Rutkowski, J. Siekmann, R. Tadeusiewicz, L.A. Zadeh (Eds.), Artificial Intelligence and Soft Computing - ICAISC 2004. XXV, 1208 pages. 2004. (Subseries LNAI).

Vol. 3068: E. André, L. Dybkj\ae r, W. Minker, P. Heisterkamp (Eds.), Affective Dialogue Systems. XII, 324 pages. 2004. (Subseries LNAI).

Vol. 3066: S. Tsumoto, R. S lowiński, J. Komorowski, J.W. Grzymala-Busse (Eds.), Rough Sets and Current Trends in Computing. XX, 853 pages. 2004. (Subseries LNAI).

Vol. 3065: A. Lomuscio, D. Nute (Eds.), Deontic Logic in Computer Science. X, 275 pages. 2004. (Subseries LNAI).

Vol. 3064: D. Bienstock, G. Nemhauser (Eds.), Integer Programming and Combinatorial Optimization. XI, 445 pages. 2004.

Vol. 3063: A. Llamosí, A. Strohmeier (Eds.), Reliable Software Technologies - Ada-Europe 2004. XIII, 333 pages. 2004.

Vol. 3062: J.L. Pfaltz, M. Nagl, B. Böhlen (Eds.), Applications of Graph Transformations with Industrial Relevance. XV, 500 pages. 2004.

Vol. 3060: A.Y. Tawfik, S.D. Goodwin (Eds.), Advances in Artificial Intelligence. XIII, 582 pages. 2004. (Subseries LNAI).

Vol. 3059: C.C. Ribeiro, S.L. Martins (Eds.), Experimental and Efficient Algorithms. X, 586 pages. 2004.

Vol. 3058: N. Sebe, M.S. Lew, T.S. Huang (Eds.), Computer Vision in Human-Computer Interaction. X, 233 pages. 2004.

Vol. 3056: H. Dai, R. Srikant, C. Zhang (Eds.), Advances in Knowledge Discovery and Data Mining. XIX, 713 pages. 2004. (Subseries LNAI).

Vol. 3054: I. Crnkovic, J.A. Stafford, H.W. Schmidt, K. Wallnau (Eds.), Component-Based Software Engineering. XI, 311 pages. 2004.

Vol. 3053: C. Bussler, J. Davies, D. Fensel, R. Studer (Eds.), The Semantic Web: Research and Applications. XIII, 490 pages. 2004.

Vol. 3052: W. Zimmermann, B. Thalheim (Eds.), Abstract State Machines 2004. Advances in Theory and Practice. XII, 235 pages. 2004.

Vol. 3051: R. Berghammer, B. Möller, G. Struth (Eds.), Relational and Kleene-Algebraic Methods in Computer Science. X, 279 pages. 2004.

Vol. 3050: J. Domingo-Ferrer, V. Torra (Eds.), Privacy in Statistical Databases. IX, 367 pages. 2004.

Vol. 3047: F. Oquendo, B. Warboys, R. Morrison (Eds.), Software Architecture. X, 279 pages. 2004.

Vol. 3046: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1016 pages. 2004.

Vol. 3045: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1040 pages. 2004.

Vol. 3044: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1140 pages. 2004.

Vol. 3043: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1180 pages. 2004.

Vol. 3042: N. Mitrou, K. Kontovasilis, G.N. Rouskas, I. Iliadis, L. Merakos (Eds.), NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications. XXXIII, 1519 pages. 2004.

Vol. 3039: M. Bubak, G.D.v. Albada, P.M. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 1271 pages. 2004.

Vol. 3038: M. Bubak, G.D.v. Albada, P.M. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 1311 pages. 2004.

Vol. 3037: M. Bubak, G.D.v. Albada, P.M. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 745 pages. 2004.

Vol. 3036: M. Bubak, G.D.v. Albada, P.M. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 713 pages. 2004.

Vol. 3035: M.A. Wimmer (Ed.), Knowledge Management in Electronic Government. XII, 326 pages. 2004. (Subseries LNAI).

Vol. 3034: J. Favela, E. Menasalvas, E. Chávez (Eds.), Advances in Web Intelligence. XIII, 227 pages. 2004. (Subseries LNAI).

Vol. 3033: M. Li, X.-H. Sun, Q. Deng, J. Ni (Eds.), Grid and Cooperative Computing. XXXVIII, 1076 pages. 2004.

Vol. 3032: M. Li, X.-H. Sun, Q. Deng, J. Ni (Eds.), Grid and Cooperative Computing. XXXVII, 1112 pages. 2004.

Vol. 3031: A. Butz, A. Krüger, P. Olivier (Eds.), Smart Graphics. X, 165 pages. 2004.

Vol. 3030: P. Giorgini, B. Henderson-Sellers, M. Winikoff (Eds.), Agent-Oriented Information Systems. XIV, 207 pages. 2004. (Subseries LNAI).

Vol. 3029: B. Orchard, C. Yang, M. Ali (Eds.), Innovations in Applied Artificial Intelligence. XXI, 1272 pages. 2004. (Subseries LNAI).

Vol. 3028: D. Neuenschwander, Probabilistic and Statistical Methods in Cryptology. X, 158 pages. 2004.

Vol. 3027: C. Cachin, J. Camenisch (Eds.), Advances in Cryptology - EUROCRYPT 2004. XI, 628 pages. 2004.

Vol. 3026: C. Ramamoorthy, R. Lee, K.W. Lee (Eds.), Software Engineering Research and Applications. XV, 377 pages. 2004.

Vol. 3025: G.A. Vouros, T. Panayiotopoulos (Eds.), Methods and Applications of Artificial Intelligence. XV, 546 pages. 2004. (Subseries LNAI).

Vol. 3024: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 621 pages. 2004.

Vol. 3023: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 611 pages. 2004.

Vol. 3022: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 621 pages. 2004.

Vol. 3021: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 633 pages. 2004.

Vol. 3019: R. Wyrzykowski, J.J. Dongarra, M. Paprzycki, J. Wasniewski (Eds.), Parallel Processing and Applied Mathematics. XIX, 1174 pages. 2004.

Vol. 3018: M. Bruynooghe (Ed.), Logic Based Program Synthesis and Transformation. X, 233 pages. 2004.

Vol. 3016: C. Lengauer, D. Batory, C. Consel, M. Odersky (Eds.), Domain-Specific Program Generation. XII, 325 pages. 2004.

Vol. 3015: C. Barakat, I. Pratt (Eds.), Passive and Active Network Measurement. XI, 300 pages. 2004.

Vol. 3014: F. van der Linden (Ed.), Software Product-Family Engineering. IX, 486 pages. 2004.

Vol. 3012: K. Kurumatani, S.-H. Chen, A. Ohuchi (Eds.), Multi-Agnets for Mass User Support. X, 217 pages. 2004. (Subseries LNAI).

Vol. 3011: J.-C. Régin, M. Rueher (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. XI, 415 pages. 2004.

Vol. 3010: K.R. Apt, F. Fages, F. Rossi, P. Szeredi, J. Váncza (Eds.), Recent Advances in Constraints. VIII, 285 pages. 2004. (Subseries LNAI).

Vol. 3009: F. Bomarius, H. Iida (Eds.), Product Focused Software Process Improvement. XIV, 584 pages. 2004.

Vol. 3008: S. Heuel, Uncertain Projective Geometry. XVII, 205 pages. 2004.

Vol. 3007: J.X. Yu, X. Lin, H. Lu, Y. Zhang (Eds.), Advanced Web Technologies and Applications. XXII, 936 pages. 2004.

Vol. 3006: M. Matsui, R. Zuccherato (Eds.), Selected Areas in Cryptography. XI, 361 pages. 2004.

Vol. 3005: G.R. Raidl, S. Cagnoni, J. Branke, D.W. Corne, R. Drechsler, Y. Jin, C.G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G.D. Smith, G. Squillero (Eds.), Applications of Evolutionary Computing. XVII, 562 pages. 2004.

Vol. 3004: J. Gottlieb, G.R. Raidl (Eds.), Evolutionary Computation in Combinatorial Optimization. X, 241 pages. 2004.

Vol. 3003: M. Keijzer, U.-M. O'Reilly, S.M. Lucas, E. Costa, T. Soule (Eds.), Genetic Programming. XI, 410 pages. 2004.

Vol. 3002: D.L. Hicks (Ed.), Metainformatics. X, 213 pages. 2004.

Vol. 3001: A. Ferscha, F. Mattern (Eds.), Pervasive Computing. XVII, 358 pages. 2004.

Vol. 2999: E.A. Boiten, J. Derrick, G. Smith (Eds.), Integrated Formal Methods. XI, 541 pages. 2004.

Vol. 2998: Y. Kameyama, P.J. Stuckey (Eds.), Functional and Logic Programming. X, 307 pages. 2004.

Vol. 2997: S. McDonald, J. Tait (Eds.), Advances in Information Retrieval. XIII, 427 pages. 2004.

Vol. 2996: V. Diekert, M. Habib (Eds.), STACS 2004. XVI, 658 pages. 2004.

Vol. 2995: C. Jensen, S. Poslad, T. Dimitrakos (Eds.), Trust Management. XIII, 377 pages. 2004.

Vol. 2994: E. Rahm (Ed.), Data Integration in the Life Sciences. X, 221 pages. 2004. (Subseries LNBI).

Vol. 2993: R. Alur, G.J. Pappas (Eds.), Hybrid Systems: Computation and Control. XII, 674 pages. 2004.

Vol. 2992: E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, E. Ferrari (Eds.), Advances in Database Technology - EDBT 2004. XVIII, 877 pages. 2004.

Vol. 2991: R. Alt, A. Frommer, R.B. Kearfott, W. Luther (Eds.), Numerical Software with Result Verification. X, 315 pages. 2004.

Vol. 2990: J. Leite, A. Omicini, L. Sterling, P. Torroni (Eds.), Declarative Agent Languages and Technologies. XII, 281 pages. 2004. (Subseries LNAI).

Vol. 2989: S. Graf, L. Mounier (Eds.), Model Checking Software. X, 309 pages. 2004.

Vol. 2988: K. Jensen, A. Podelski (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XIV, 608 pages. 2004.

Preface

This volume contains selected papers from LOPSTR 2003, the 13th International Symposium on Logic-Based Program Synthesis and Transformation. The LOPSTR series is devoted to research in logic-based program development. Particular topics of interest are specification, synthesis, verification, transformation, specialization, analysis, optimization, composition, reuse, component-based software development, agent-based software development, software architectures, design patterns and frameworks, program refinement and logics for refinement, proofs as programs, and applications and tools.

LOPSTR 2003 took place at the University of Uppsala from August 25 to August 27 as part of PLI 2003 (Principles, Logics, and Implementations of High-Level Programming Languages). PLI was an ACM-organized confederation of conferences and workshops with ICFP 2003 (ACM-SIGPLAN International Conference on Functional Programming) and PPDP 2003 (ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming) as the main events. The LOPSTR community profited from the shared lectures of the invited speakers, and the active scientific discussions enabled by the co-location.

LOPSTR 2003 was the thirteenth in a series of events. Past events were held in Manchester, UK (1991, 1992, 1998), Louvain-la-Neuve, Belgium (1993), Pisa, Italy (1994), Arnhem, The Netherlands (1995), Stockholm, Sweden (1996), Leuven, Belgium (1997), Venice, Italy (1999), London, UK (2000), Paphos, Cyprus (2001), and Madrid, Spain (2002).

I wish to thank the PLI Organizing Committee and especially Kostis Sagonas for welcoming LOPSTR as part of PLI 2003 and taking care of the many organizational matters. Special thanks go towards Roland Bol for taking care of LOPSTR specific matters in Uppsala, towards Wim Vanhoof for assistance with the Program Chair work, and towards Qiang Fu for the help in preparing this volume. The sponsorship of the Association for Logic Programming (ALP) is gratefully acknowledged, and the LNCS team of Springer-Verlag is thanked for publishing this volume with the selected and revised papers. Last but not least, authors, PC members and additional reviewers are thanked for all the work that went into preparing this selection of revised papers.

Out of 32 submissions, the program committee selected 18 works for presentation; 16 were revised and submitted for this volume. The program committee selected 12 of them for inclusion in this volume. In addition, a paper based on the invited talk of Michael Leuschel is included as well as some abstracts based on the other papers presented at LOPSTR.

The preproceedings were printed in Uppsala and are also available at http://www.cs.kuleuven.ac.be/ dtai/lopstr03/, a page with all information about LOPSTR 2003.

March 2004

Maurice Bruynooghe

Program Chair

Maurice Bruynooghe	Katholieke Universiteit Leuven, Belgium

Local Chair

Roland Bol	Uppsala University, Sweden

Program Committee

Elvira Albert	Complutense University of Madrid, Spain
Roland Bol	Uppsala University, Sweden
Maurice Bruynooghe	Katholieke Universiteit Leuven, Belgium
Michael Butler	University of Southampton, UK
Jim Caldwell	University of Wyoming, USA
Włodek Drabent	Polish Academy of Sciences, Poland
	Linköping University, Sweden
Tom Ellman	Vassar College, USA
Norbert E. Fuchs	University of Zurich, Switzerland
Robert Glück	Waseda University, Japan
Gopal Gupta	University of Texas at Dallas, USA
Ian Hayes	University of Queensland, Australia
Catholijn Jonker	Vrije Universiteit Amsterdam, The Netherlands
Andy King	University of Kent, UK
Mario Ornaghi	Universitá degli Studi di Milano, Italy
Maurizio Proietti	IASI-CNR, Rome, Italy
Germán Puebla	Technical University of Madrid, Spain
Julian Richardson	RIACS/NASA Ames Research Center, USA
Olivier Ridoux	University of Rennes 1/IRISA, France
Sabina Rossi	Università Ca' Foscari di Venezia, Italy
Wim Vanhoof	University of Namur, Belgium

Local Organizers

Roland Bol
Kostis Sagonas

Additional Referees

José Alferes
Olaf Chitil
Nicoletta Cocco
Dario Colazzo
John Cowles
Agostino Dovier
Thomas Eiter
Samir Genaim
Silvio Ghilardi
Roberta Gori
Stefan Gruner
Hai-Feng Guo
Ángel Herranz
Juliana Küster-Filipe
Kung-Kiu Lau
Lunjin Lu
Alberto Momigliano
Alberto Pettorossi
Isabel Pita
Enrico Pontelli
Alessandra Raffaetá
Konstantinos Sagonas
Alexander Serebrenik
Jan-Georg Smaus
Colin Snook
Vera Stebletsova
Bert Van Nuffelen
Pedro Vasconcelos
Germán Vidal

Table of Contents

Invited Talk

Inductive Theorem Proving by Program Specialisation:
Generating Proofs for Isabelle Using Ecce 1
Helko Lehmann and Michael Leuschel

Specification and Synthesis

Predicate Synthesis from Inductive Proof Attempt of Faulty Conjectures .. 20
Francis Alexandre, Khaled Bsaïes, and Moussa Demba

Correct OO Systems in Computational Logic 34
Kung-Kiu Lau and Mario Ornaghi

Specification and Synthesis of Hybrid Automata
for Physics-Based Animation 54
Thomas Ellman

Adding Concrete Syntax to a Prolog-Based Program Synthesis System
(Extended Abstract) 56
Bernd Fischer and Eelco Visser

Verification

Formal Development and Verification of Approximation Algorithms
Using Auxiliary Variables 59
Rudolf Berghammer and Markus Müller-Olm

Formal Reasoning about Efficient Data Structures:
A Case Study in ACL2 75
José Luis Ruiz-Reina, José Antonio Alonso-Jiménez,
María José Hidalgo, and Francisco Jesús Martín-Mateos

Analysis

A Program Transformation for Backwards Analysis of Logic Programs 92
John P. Gallagher

An Efficient Staging Algorithm for Binding-Time Analysis 106
Takuma Murakami, Zhenjiang Hu, Kazuhiko Kakehi,
and Masato Takeichi

Proving Termination with Adornments 108
Alexander Serebrenik and Danny De Schreye

Transformation and Specialisation

Constructively Characterizing Fold and Unfold 110
Tjark Weber and James Caldwell

Deterministic Higher-Order Patterns for Program Transformation 128
Tetsuo Yokoyama, Zhenjiang Hu, and Masato Takeichi

From Interpreter to Logic Engine by Defunctionalization 143
Dariusz Biernacki and Olivier Danvy

Linearization by Program Transformation 160
Sandra Alves and Mário Florido

Continuation Semantics as Horn Clauses 176
Qian Wang and Gopal Gupta

Constraints

Simplification of Database Integrity Constraints Revisited:
A Transformational Approach 178
Henning Christiansen and Davide Martinenghi

Integration and Optimization of Rule-Based Constraint Solvers 198
Slim Abdennadher and Thom Frühwirth

Introducing ESRA, a Relational Language
for Modelling Combinatorial Problems 214
Pierre Flener, Justin Pearson, and Magnus Ågren

Author Index 233

Inductive Theorem Proving by Program Specialisation: Generating Proofs for Isabelle Using Ecce

Helko Lehmann and Michael Leuschel

Department of Electronics and Computer Science
University of Southampton
Highfield, Southampton, SO17 1BJ, UK
{hel99r,mal}@ecs.soton.ac.uk

Abstract. In this paper we discuss the similarities between program specialisation and inductive theorem proving, and then show how program specialisation can be used to perform inductive theorem proving. We then study this relationship in more detail for a particular class of problems (verifying infinite state Petri nets) in order to establish a clear link between program specialisation and inductive theorem proving. In particular, we use the program specialiser ECCE to generate specifications, hypotheses and proof scripts in the theory format of the proof assistant ISABELLE. Then, in many cases, ISABELLE can automatically execute these proof scripts and thereby verify the soundness of ECCE's verification process and of the correspondence between program specialisation and inductive theorem proving.

1 Introduction

Program specialisation aims at improving the overall performance of programs by performing source to source transformations. A common approach, known as partial evaluation [8], is to exploit partial knowledge about the input by precomputing parts of the program. In the context of logic programming, partial evaluation is sometimes called partial deduction and is achieved through a well-automated application of parts of the Burstall-Darlington unfold/fold transformation framework.

The relation between program specialisation and theorem proving has already been raised several times in the literature [23, 7, 24, 21]. In this paper we will examine in closer detail the relationship between partial deduction and inductive theorem proving.

Partial Deduction. At the heart of any technique for *partial deduction* is a program analysis phase: Given a program P and an (atomic) goal $\leftarrow A$, one aims to analyse the computation-flow of P for all instances $\leftarrow A\theta$ of $\leftarrow A$. Based on the results of this analysis, new program clauses are synthesised.

M. Bruynooghe (Ed.): LOPSTR 2004, LNCS 3018, pp. 1–19, 2004.

In partial deduction, such an analysis is based on the construction of finite and usually incomplete[1], SLD(NF)-trees. More specifically, following the foundations for partial deduction developed in [17] (see also [12] for an up-to-date overview), one constructs

- a finite set of atoms $S = \{A_1, \ldots, A_n\}$, and
- a finite (possibly incomplete) SLD(NF)-tree τ_i for each $(P \cup \{\leftarrow A_i\})$,

such that:

1) the atom A in the initial goal $\leftarrow A$ is an instance of some A_i in S, and
2) for each goal $\leftarrow B_1, \ldots, B_k$ labelling a leaf of some SLD(NF)-tree τ_l, each B_i is an instance of some A_j in S.

The construction of the set S is referred to as the *global control*, while the construction of the trees τ_i are called the *local control*. The conditions 1) and 2) are referred to as *closedness* and ensure that *together* the SLD(NF)-trees $\tau_1, \ldots, \tau_n$ form a complete description of all possible computations that can occur for all concrete instances $\leftarrow A\theta$ of the goal of interest. Finally, a code generation phase produces a *resultant clause* for each non-failing branch of each tree, which synthesises the computation in that branch. This phase also typically generates a fresh predicate name for every element of the set S and rename the clauses in an appropriate manner.

The approach has been generalised to specialising a set of *conjunctions* rather than just atoms in [4]. The basic principle remains roughly as outlined above; the only difference being that we have a set S of conjunctions rather than atoms and that the closedness condition becomes slightly more involved to allow the leaf goals $\leftarrow B_1, \ldots, B_k$ to be split up into sub-conjunctions. This technique has been implemented within the program specialiser ECCE [15, 4] . An overview of control techniques that are used in partial deduction and conjunctive partial deduction in general and by ECCE in particular, such as determinacy, homeomorphic embedding, or characteristic trees, can be found in [12].

A Small Example. Let us illustrate conjunctive partial deduction on the following simple program.

```
even(0).
even(s(X)) :- odd(X).
odd(s(X)) :- even(X).
```

Suppose we only wish to use this program for queries of the form $\leftarrow C$ with $C = even(X) \wedge odd(X)$ Conjunctive partial deduction can then specialise this program by constructing the incomplete SLD-tree for $\leftarrow C$ depicted in

[1] As usual in partial deduction, we assume that the notion of an SLD-tree is generalised [17] to allow it to be incomplete: at any point we may decide not to select any atom and terminate a derivation.

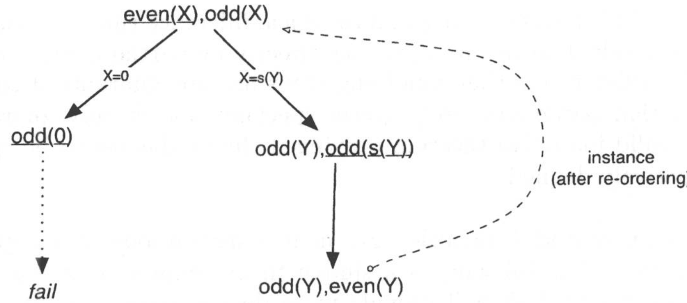

Fig. 1. Specialisation of even-odd

Fig. 1. The set S mentioned above would simply be $S = \{C\}$. Supposing that we produce the new predicate name **even_odd** for C, the specialised program we obtain, is:

```
even_odd(s(X)) :- even_odd(X).
```

It is immediately obvious that $even\_odd(X)$ will never succeed, and hence that no number is even and odd at the same time. The ECCE system [15, 4] basically produces the above result[2] and can also automatically infer the failure of $even\_odd(X)$ by applying its bottom up more specific program construction phase [18] in the post-processing.

Inductive Theorem Proving. Now, the above result corresponds to an inductive proof showing that no number can be both even and odd. The left branch of Fig. 1 corresponds to examining the base case $X = 0$, while the right branch corresponds to the induction step whereby $even(s(Y)), odd(s(Y))$ is rewritten into the equivalent $odd(Y), even(Y)$ so that the induction hypothesis can be applied.

In a sense the conjunctive partial deduction has identified a working induction schema and the bottom-up propagation [18] has performed the induction proper. This highlights a similarity between partial deduction and *inductive theorem proving*. Indeed, in the induction step of an inductive proof one tries to transform the induction assumption(s) for $n+1$ using basic inference rules so as to be able to apply the induction hypothese(s) and complete the proof. In partial deduction, one tries to transform the atoms in A (or conjunctions for conjunctive partial deduction) by unfolding so as to be able to "fold" back all leaves. The set of atoms A thus plays the role of the induction hypotheses and resolution the role of classical theorem proving steps. In summary,

- there is a striking similarity between the control problems of partial deduction and inductive theorem proving. The problem of ensuring A-closedness is basically the same as finding induction hypotheses where the induction "goes through." Many control techniques have been developed in either camp (e.g., [1] for inductive theorem proving) and cross-fertilisation might be possible.

[2] Using the default settings, ECCE produces a slightly bigger specialised program because it does not re-order atoms by default. But the overall result is the same.

– if basic resolution steps correspond to logical inference rules one may be able to perform inductive theorem proving directly by partial deduction.
  The only difference is that unfolding steps are not guaranteed to decrease the induction parameter, so program specialisation is only guaranteed to perform valid inductive theorem proving if the predicates to be specialised are inductively defined.

A More Complicated Example. Let us now have a look at a slightly more involved example. The following is a simple theory expressed in the proof assistant ISABELLE [19]. (We will provide more details about ISABELLE later in the paper.) The theory defines a datatype for binary trees and then defines the function mirror which simply produces the mirror image of tree (i.e., reversing left and right children for all nodes). We then define a lemma stating that applying mirror twice produces the same result and then instruct Isabelle to use induction on the tree in order to show this lemma.

```
theory ToyTree = PreList:
  datatype 'a tree = Tip                  ("[]")
                   | Node "'a tree" 'a "'a tree"
  consts mirror :: "'a tree => 'a tree"
  primrec
    "mirror([]) = []"
    "mirror((Node ls x rs)) = Node (mirror(rs)) x (mirror(ls))"
lemma mirror_mirror [simp]: "mirror(mirror(xs)) = xs"
apply (induct_tac xs)
```

Loading this theory int ISABELLE results in the following output:

```
proof (prove): step 1
fixed variables: xs

goal (lemma (mirror_mirror), 2 subgoals):
 1. mirror (mirror []) = []
 2. !!tree1 a tree2.
       [| mirror (mirror tree1) = tree1; mirror (mirror tree2) = tree2 |]
       ==> mirror (mirror (Node tree1 a tree2)) = Node tree1 a tree2
```

It is now possible to use ISABELLE to prove this lemma, by interactively performing the required rewriting steps and twice applying the induction hypothesis[3].

Let us now try to achieve the same result using program specialisation. First, we have to encode the mirror function and the lemma as a logic program:

```
mirror(tip,tip).
mirror(tree(L,N,R),tree(RR,N,RL)) :- mirror(L,RL), mirror(R,RR).
lemma(X,R) :- mirror(X,Z),mirror(Z,R).
```

[3] E.g., first calling the simplifier `apply(simp)` and then the automatic prover `apply(auto)` will perform the required proof steps.

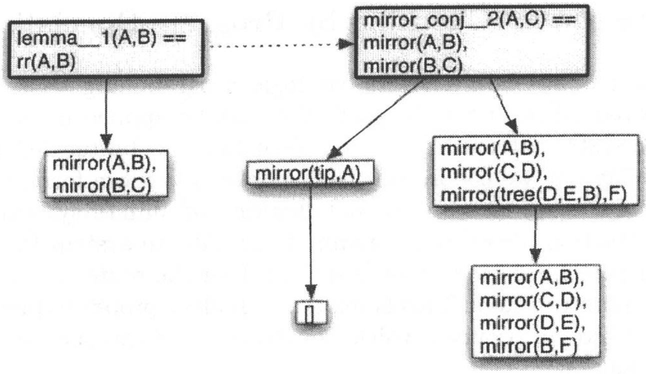

Fig. 2. ECCE specialisation tree for mirror

Now, one would like to be able to infer that for all valid trees the the second argument of `lemma` must be identical to the first argument. Surprisingly this is exactly what we obtain when we specialise the above program for the call `lemma(X,R)` using the ECCE program specialiser (with the most specific version [18] postprocessing enabled):

```
/* Transformation time: 130 ms */
/* Specialised Predicates:
lemma__1(A,B) :- lemma(A,B).
mirror_conj__2(A,B) :- mirror(A,C1), mirror(C1,B). */

lemma(A,A) :- mirror_conj__2(A,A).
lemma__1(A,A) :-  mirror_conj__2(A,A).
mirror_conj__2(tip,tip).
mirror_conj__2(tree(A,B,C),tree(A,B,C)) :-
    mirror_conj__2(A,A),  mirror_conj__2(C,C).
```

Again, ECCE has managed to rewrite the lemma in such a way that the induction hypothesis could be applied (in this case it was applied twice as can be seen from the two instances of `mirror_conj__2` in the last clause of the specialised program). The specialisation tree produced by ECCE can be seen in Fig. 2. The dashed arrows indicate a descendance at the global control level (see, e.g., [12]), whereas the solid arrows indicate unfolding steps. By carefully inspecting the proof trace of ISABELLE and the specialisation tree of ECCE it turns out that there is a one-to-one correspondence between the steps performed by Isabelle and by ECCE.

An obvious question is now whether there is a systematic way to exploit this correspondence? In the next sections we show how ECCE can be used to perform inductive theorem proving as applied to verification tasks and how the specialisation trees produced by ECCE can be automatically translated into induction schemas for the proof assistant Isabelle [19].