Klaus Schneider
Jens Brandt (Eds.)

LNCS 4732

Theorem Proving in Higher Order Logics

20th International Conference, TPHOLs 2007
Kaiserslautern, Germany, September 2007
Proceedings

Springer

Klaus Schneider Jens Brandt (Eds.)

Theorem Proving in Higher Order Logics

20th International Conference, TPHOLs 2007
Kaiserslautern, Germany, September 10-13, 2007
Proceedings

Springer

Volume Editors

Klaus Schneider
Jens Brandt
University of Kaiserslautern
Department of Computer Science
Reactive Systems Group
P.O.Box 3049, 67653 Kaiserslautern, Germany
E-mail: {klaus.schneider,brandt}@informatik.uni-kl.de

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.4.1, I.2.3, F.3.1, D.2.4, B.6.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-74590-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-74590-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12115440 06/3180 5 4 3 2 1 0

Lecture Notes in Computer Science 4732

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Lecture Notes in Computer Science

Sublibrary 1: Theoretical Computer Science and General Issues

For information about Vols. 1– 4421
please contact your bookseller or Springer

Vol. 4743: P. Thulasiraman, X. He, T.L. Xu, M.K. Denko, R.K. Thulasiram, L.T. Yang (Eds.), Frontiers of High Performance Computing and Networking ISPA 2007 Workshops. XXIX, 536 pages. 2007.

Vol. 4742: I. Stojmenovic, R.K. Thulasiram, L.T. Yang, W. Jia, M. Guo, R.F. de Mello (Eds.), Parallel and Distributed Processing and Applications. XX, 995 pages. 2007.

Vol. 4736: S. Winter, M. Duckham, L. Kulik, B. Kuipers (Eds.), Spatial Information Theory. XV, 455 pages. 2007.

Vol. 4732: K. Schneider, J. Brandt (Eds.), Theorem Proving in Higher Order Logics. IX, 401 pages. 2007.

Vol. 4708: L. Kučera, A. Kučera (Eds.), Mathematical Foundations of Computer Science 2007. XVIII, 764 pages. 2007.

Vol. 4707: O. Gervasi, M.L. Gavrilova (Eds.), Computational Science and Its Applications – ICCSA 2007, Part III. XXIV, 1205 pages. 2007.

Vol. 4706: O. Gervasi, M.L. Gavrilova (Eds.), Computational Science and Its Applications – ICCSA 2007, Part II. XXIII, 1129 pages. 2007.

Vol. 4705: O. Gervasi, M.L. Gavrilova (Eds.), Computational Science and Its Applications – ICCSA 2007, Part I. XLIV, 1169 pages. 2007.

Vol. 4703: L. Caires, V.T. Vasconcelos (Eds.), CONCUR 2007 – Concurrency Theory. XIII, 507 pages. 2007.

Vol. 4697: L. Choi, Y. Paek, S. Cho (Eds.), Advances in Computer Systems Architecture. XIII, 400 pages. 2007.

Vol. 4688: K. Li, M. Fei, G.W. Irwin, S. Ma (Eds.), Bio-Inspired Computational Intelligence and Applications. XIX, 805 pages. 2007.

Vol. 4684: L. Kang, Y. Liu, S. Zeng (Eds.), Evolvable Systems: From Biology to Hardware. XIV, 446 pages. 2007.

Vol. 4683: L. Kang, Y. Liu, S. Zeng (Eds.), Intelligence Computation and Applications. XVII, 663 pages. 2007.

Vol. 4681: D.-S. Huang, L. Heutte, M. Loog (Eds.), Advanced Intelligent Computing Theories and Applications. XXVI, 1379 pages. 2007.

Vol. 4671: V. Malyshkin (Ed.), Parallel Computing Technologies. XIV, 635 pages. 2007.

Vol. 4668: J.M. de Sá, L.A. Alexandre, W. Duch, D.P. Mandic (Eds.), Artificial Neural Networks – ICANN 2007, Part I. XXXI, 978 pages. 2007.

Vol. 4667: J. Hertzberg, M. Beetz, R. Englert (Eds.), KI 2007: Advances in Artificial Intelligence. IX, 516 pages. 2007.

Vol. 4666: M.E. Davies, C.J. James, S.A. Abdallah, M.D Plumbley (Eds.), Independent Component Analysis and Blind Signal Separation. XIX, 847 pages. 2007.

Vol. 4664: J. Durand-Lose, M. Margenstern (Eds.), Machines, Computations, and Universality. X, 325 pages. 2007.

Vol. 4649: V. Diekert, M.V. Volkov, A. Voronkov (Eds.), Computer Science – Theory and Applications. XIII, 420 pages. 2007.

Vol. 4647: R. Martin, M. Sabin, J. Winkler (Eds.), Mathematics of Surfaces XII. IX, 509 pages. 2007.

Vol. 4644: N. Azemard, L. Svensson (Eds.), Integrated Circuit and System Design. XIV, 583 pages. 2007.

Vol. 4641: A.-M. Kermarrec, L. Bougé, T. Priol (Eds.), Euro-Par 2007 Parallel Processing. XXVII, 974 pages. 2007.

Vol. 4639: E. Csuhaj-Varjú, Z. Ésik (Eds.), Fundamentals of Computation Theory. XIV, 508 pages. 2007.

Vol. 4638: T. Stützle, M. Birattari, H.H. Hoos (Eds.), Engineering Stochastic Local Search Algorithms. X, 223 pages. 2007.

Vol. 4628: L.N. de Castro, F.J. Von Zuben, H. Knidel (Eds.), Artificial Immune Systems. XII, 438 pages. 2007.

Vol. 4627: M. Charikar, K. Jansen, O. Reingold, J.D.P. Rolim (Eds.), Approximation, Randomization, and Combinatorial Optimization. XII, 626 pages. 2007.

Vol. 4624: T. Mossakowski, U. Montanari, M. Haveraaen (Eds.), Algebra and Coalgebra in Computer Science. XI, 463 pages. 2007.

Vol. 4619: F. Dehne, J.-R. Sack, N. Zeh (Eds.), Algorithms and Data Structures. XVI, 662 pages. 2007.

Vol. 4618: S.G. Akl, C.S. Calude, M.J. Dinneen, G. Rozenberg, H.T. Wareham (Eds.), Unconventional Computation. X, 243 pages. 2007.

Vol. 4616: A. Dress, Y. Xu, B. Zhu (Eds.), Combinatorial Optimization and Applications. XI, 390 pages. 2007.

Vol. 4613: F.P. Preparata, Q. Fang (Eds.), Frontiers in Algorithmics. XI, 348 pages. 2007.

Vol. 4600: H. Comon-Lundh, C. Kirchner, H. Kirchner (Eds.), Rewriting, Computation and Proof. XVI, 273 pages. 2007.

Vol. 4599: S. Vassiliadis, M. Berekovic, T.D. Hämäläinen (Eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation. XVIII, 466 pages. 2007.

Vol. 4598: G. Lin (Ed.), Computing and Combinatorics. XII, 570 pages. 2007.

Vol. 4596: L. Arge, C. Cachin, T. Jurdziński, A. Tarlecki (Eds.), Automata, Languages and Programming. XVII, 953 pages. 2007.

Vol. 4595: D. Bošnački, S. Edelkamp (Eds.), Model Checking Software. X, 285 pages. 2007.

Vol. 4590: W. Damm, H. Hermanns (Eds.), Computer Aided Verification. XV, 562 pages. 2007.

Vol. 4588: T. Harju, J. Karhumäki, A. Lepistö (Eds.), Developments in Language Theory. XI, 423 pages. 2007.

Vol. 4583: S.R. Della Rocca (Ed.), Typed Lambda Calculi and Applications. X, 397 pages. 2007.

Vol. 4580: B. Ma, K. Zhang (Eds.), Combinatorial Pattern Matching. XII, 366 pages. 2007.

Vol. 4576: D. Leivant, R. de Queiroz (Eds.), Logic, Language, Information and Computation. X, 363 pages. 2007.

Vol. 4547: C. Carlet, B. Sunar (Eds.), Arithmetic of Finite Fields. XI, 355 pages. 2007.

Vol. 4546: J. Kleijn, A. Yakovlev (Eds.), Petri Nets and Other Models of Concurrency – ICATPN 2007. XI, 515 pages. 2007.

Vol. 4545: H. Anai, K. Horimoto, T. Kutsia (Eds.), Algebraic Biology. XIII, 379 pages. 2007.

Vol. 4533: F. Baader (Ed.), Term Rewriting and Applications. XII, 419 pages. 2007.

Vol. 4528: J. Mira, J.R. Álvarez (Eds.), Nature Inspired Problem-Solving Methods in Knowledge Engineering, Part II. XXII, 650 pages. 2007.

Vol. 4527: J. Mira, J.R. Álvarez (Eds.), Bio-inspired Modeling of Cognitive Tasks, Part I. XXII, 630 pages. 2007.

Vol. 4525: C. Demetrescu (Ed.), Experimental Algorithms. XIII, 448 pages. 2007.

Vol. 4514: S.N. Artemov, A. Nerode (Eds.), Logical Foundations of Computer Science. XI, 513 pages. 2007.

Vol. 4513: M. Fischetti, D.P. Williamson (Eds.), Integer Programming and Combinatorial Optimization. IX, 500 pages. 2007.

Vol. 4510: P. Van Hentenryck, L.A. Wolsey (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. X, 391 pages. 2007.

Vol. 4507: F. Sandoval, A. Prieto, J. Cabestany, M. Graña (Eds.), Computational and Ambient Intelligence. XXVI, 1167 pages. 2007.

Vol. 4501: J. Marques-Silva, K.A. Sakallah (Eds.), Theory and Applications of Satisfiability Testing – SAT 2007. XI, 384 pages. 2007.

Vol. 4497: S.B. Cooper, B. Löwe, A. Sorbi (Eds.), Computation and Logic in the Real World. XVIII, 826 pages. 2007.

Vol. 4494: H. Jin, O.F. Rana, Y. Pan, V.K. Prasanna (Eds.), Algorithms and Architectures for Parallel Processing. XIV, 508 pages. 2007.

Vol. 4493: D. Liu, S. Fei, Z. Hou, H. Zhang, C. Sun (Eds.), Advances in Neural Networks – ISNN 2007, Part III. XXVI, 1215 pages. 2007.

Vol. 4492: D. Liu, S. Fei, Z. Hou, H. Zhang, C. Sun (Eds.), Advances in Neural Networks – ISNN 2007, Part II. XXVII, 1321 pages. 2007.

Vol. 4491: D. Liu, S. Fei, Z.-G. Hou, H. Zhang, C. Sun (Eds.), Advances in Neural Networks – ISNN 2007, Part I. LIV, 1365 pages. 2007.

Vol. 4490: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Part IV. XXXVII, 1211 pages. 2007.

Vol. 4489: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Part III. XXXVII, 1257 pages. 2007.

Vol. 4488: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Part II. XXXV, 1251 pages. 2007.

Vol. 4487: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Part I. LXXXI, 1275 pages. 2007.

Vol. 4484: J.-Y. Cai, S.B. Cooper, H. Zhu (Eds.), Theory and Applications of Models of Computation. XIII, 772 pages. 2007.

Vol. 4475: P. Crescenzi, G. Prencipe, G. Pucci (Eds.), Fun with Algorithms. X, 273 pages. 2007.

Vol. 4474: G. Prencipe, S. Zaks (Eds.), Structural Information and Communication Complexity. XI, 342 pages. 2007.

Vol. 4459: C. Cérin, K.-C. Li (Eds.), Advances in Grid and Pervasive Computing. XVI, 759 pages. 2007.

Vol. 4449: Z. Horváth, V. Zsók, A. Butterfield (Eds.), Implementation and Application of Functional Languages. X, 271 pages. 2007.

Vol. 4448: M. Giacobini (Ed.), Applications of Evolutionary Computing. XXIII, 755 pages. 2007.

Vol. 4447: E. Marchiori, J.H. Moore, J.C. Rajapakse (Eds.), Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. XI, 302 pages. 2007.

Vol. 4446: C. Cotta, J. van Hemert (Eds.), Evolutionary Computation in Combinatorial Optimization. XII, 241 pages. 2007.

Vol. 4445: M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A.I. Esparcia-Alcázar (Eds.), Genetic Programming. XI, 382 pages. 2007.

Vol. 4436: C.R. Stephens, M. Toussaint, D. Whitley, P.F. Stadler (Eds.), Foundations of Genetic Algorithms. IX, 213 pages. 2007.

Vol. 4433: E. Şahin, W.M. Spears, A.F.T. Winfield (Eds.), Swarm Robotics. XII, 221 pages. 2007.

Vol. 4432: B. Beliczynski, A. Dzielinski, M. Iwanowski, B. Ribeiro (Eds.), Adaptive and Natural Computing Algorithms, Part II. XXVI, 761 pages. 2007.

Vol. 4431: B. Beliczynski, A. Dzielinski, M. Iwanowski, B. Ribeiro (Eds.), Adaptive and Natural Computing Algorithms, Part I. XXV, 851 pages. 2007.

Vol. 4424: O. Grumberg, M. Huth (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XX, 738 pages. 2007.

Vol. 4423: H. Seidl (Ed.), Foundations of Software Science and Computational Structures. XVI, 379 pages. 2007.

Vol. 4422: M.B. Dwyer, A. Lopes (Eds.), Fundamental Approaches to Software Engineering. XV, 440 pages. 2007.

Preface

This volume constitutes the proceedings of the 20th International Conference on Theorem Proving in Higher-Order Logics (TPHOLs 2007) held September 10–13, 2007 in Kaiserslautern, Germany. TPHOLs covers all aspects of theorem proving in higher-order logics as well as related topics in theorem proving and verification.

There were 52 submissions, and each submission was refereed by at least 4 reviewers, who had been selected by the program committee. Of these submissions, 26 were accepted for presentation at the conference and publication in this volume. In keeping with tradition, TPHOLs 2007 also offered a venue for the presentation of work in progress, where researchers invite discussion by means of a brief preliminary talk and then discuss their work at a poster session. A supplementary proceedings containing associated papers for work in progress was published by the University of Kaiserslautern. The organizers are grateful to Constance Heitmeyer (Naval Research Laboratory), Xavier Leroy (INRIA Rocquencourt) and Peter Liggesmeyer (Fraunhofer IESE) for agreeing to give invited talks at TPHOLs 2007.

The TPHOLs conference traditionally changes continent each year in order to maximize the chances of researchers from around the world being able to attend. Starting in 1993, the proceedings of TPHOLs and its predecessor workshops have been published in the Lecture Notes in Computer Science series of Springer-Verlag:

1993 Vancouver	LNCS 780	2000 Portland	LNCS 1869
1994 Valletta	LNCS 859	2001 Edinburgh	LNCS 2152
1995 Aspen Grove	LNCS 971	2002 Hampton	LNCS 2410
1996 Turku	LNCS 1125	2003 Rome	LNCS 2758
1997 Murray Hill	LNCS 1275	2004 Park City	LNCS 3223
1998 Canberra	LNCS 1479	2005 Oxford	LNCS 3603
1999 Nice	LNCS 1690	2006 Seattle	LNCS 4130

We would like to thank our sponsors: Fraunhofer IESE (Institute of Experimental Software Engineering), DASMOD (Dependable Adaptive Systems and Mathematical Modeling) Cluster, and DFKI (German Research Center for Artificial Intelligence).

July 2007

Klaus Schneider
Jens Brandt

Conference Organization

Program Chairs

Klaus Schneider
Jens Brandt

Program Committee

Mark Aagaard
Yves Bertot
Ching-Tsun Chou
Thierry Coquand
Amy Felty
Jean-Christophe Filliatre
Ganesh Gopalakrishnan
Mike Gordon
Jim Grundy
Elsa Gunter
John Harrison
Jason Hickey
Peter Homeier
Joe Hurd
Paul Jackson
Thomas Kropf
John Matthews
Tom Melham
Cesar Munoz
Tobias Nipkow
Sam Owre
Christine Paulin-Mohring
Lawrence Paulson
Klaus Schneider
Konrad Slind
Sofiene Tahar
Burkhart Wolff

External Reviewers

Behzad Akbarpour
Ulrich Berger
Stefan Berghofer
Pierre Castéran
Pierre Corbineau
Amjad Gawanmeh
Florian Haftmann
Osman Hasan
Nathan Linger
Claude Marche
Jia Meng
Paul Miner
John O'Leary
Sam Owre
Tom Ridge
Norbert Schirmer
Natarajan Shankar
Alan Smaill
Mark-Oliver Stehr
Christian Urban
Makarius Wenzel
Yu Yang
Mohamed Zaki

Table of Contents

On the Utility of Formal Methods in the Development and Certification of Software

Constance L. Heitmeyer

Naval Research Laboratory
Washington, DC 20375
http://chacs.nrl.navy.mil/personnel/heitmeyer.html

During the past three decades, many formal methods have been proposed whose goal is to improve the quality of computer systems. I use the term *formal method* to refer to any mathematically-based technique or tool useful in either hardware or software development. Recently, formal methods have played a significantly increased role in hardware design. More and more companies that sell microprocessors and hardware chips, including Intel, IBM, and Motorola, are using formally-based tools, such as model checkers, theorem provers, and equivalence checkers, to check hardware designs for flaws. While applied less frequently in practical software development, formal methods have, in a few recent cases, also been effective in detecting software defects. A prominent example is the set of tools developed in Microsoft's SLAM project which were designed to detect flaws in device drivers [1], a primary source of software defects in Microsoft programs. In 2006, Microsoft released the Static Driver Verifier (SDV) as part of Windows Vista, the latest Microsoft operating system. SDV uses the SLAM model checker to detect cases in which device drivers linked to Vista violate one of a set of interface rules.

This talk reviews several formally-based techniques of value in developing software systems, focusing on techniques for specifying, validating, and verifying software requirements [2], a primary cause of software defects. Next, the talk describes our recent experience applying formal techniques in the certification of a security-critical module of an embedded software device [3]. TAME [4], one formal technique applied in this effort, is a front-end to the higher-order logic theorem prover PVS. The benefits of using a higher-order logic are described.

References

1. Ball, T., Bounimova, E., Cook, B., Levin, V., Lichtenberg, J., McGarvey, C., Ondrusek, B., Rajamani, S., Ustuner, A.: Thorough static analysis of device drivers. In: European Systems Conference (2006)
2. Heitmeyer, C., Archer, M., Bharadwaj, R., Jeffords, R.: Tools for constructing requirements specifications: The SCR toolset at the age of ten. Computer Systems Science and Engineering 20(1), 19–35 (2005)

K. Schneider and J. Brandt (Eds.): TPHOLs 2007, LNCS 4732, pp. 1–2, 2007.

3. Heitmeyer, C.L., Archer, M., Leonard, E.I., McLean, J.: Formal specification and verification of data separation in a separation kernel for an embedded system. In: Proc. 13th ACM Conference on Computer and Communications Security, ACM Press, New York (2006)
4. Archer, M.: TAME: Using PVS strategies for special-purpose theorem proving. Annals of Mathematics and Artificial Intelligence 29(1-4), 131–189 (2001)

Formal Techniques in Software Engineering: Correct Software and Safe Systems

Peter Liggesmeyer

Department of Computer Science
University of Kaiserslautern, Germany
Peter.Liggesmeyer@informatik.uni-kl.de
Fraunhofer Institute Experimental Software Engineering
Kaiserslautern, Germany
Peter.Liggesmeyer@iese.fraunhofer.de

In embedded systems, safety and reliability are usually important quality characteristics. It is required to determine these properties including hardware and software. Many techniques have been proposed to analyze, model and predict software and hardware quality characteristics on a quantified basis, e.g. fault trees, Markov analysis, and statistical reliability models.

Formal techniques are increasingly used to prove properties of critical systems. They support safety and reliability modelling by generating models based on formal analysis. Approaches for the automated generation of fault trees augment the traditional manual procedures.

We developed fault tree generation techniques that are based on finite state descriptions, and specifications of safety properties using temporal logic. Model checking is used to determine how specific failures can cause unsafe behaviour. This information is converted into a fault tree that propagates the failure probabilities of components, e.g. sensors, on residual risks on the system level. This is a combination of formal techniques, safety modelling and statistical analysis. Finite state machines are used to represent a system comprising a controller and a technical process under control. The controller is represented by a deterministic state machine. The process under control is, in general, non-deterministic, and so is the model. In the beginning, the verification by symbolic model checking assumes that the process may produce any arbitrary input for the controller. This will in most cases yield unreasonable inputs to the controller. By means of a process specification that is used by a model checker, transitions of the process are restricted to those transitions that are physically reasonable. Model checking is then used to determine whether unsafe states are reachable if certain failures occur.

Statistical reliability growth models are another approach that may be used to measure and predict reliability and safety. Since different software reliability models can produce very different answers when used to predict future reliability, users need to know which, if any, of the competing models are trustworty in a specific context. We developed a reliability assessment tool that helps in reaching such decisions and supports the reliability analysis of software-based systems. It incorporates reliability models and supports model selection based on observed failure data using statistically sound criteria. The tool was used to

K. Schneider and J. Brandt (Eds.): TPHOLs 2007, LNCS 4732, pp. 3–4, 2007.

apply statistical reliability modelling to various projects within Siemens. These include, e.g., telecommunication software, railway systems, and medical applications. Although it only supports software reliability models, we also applied the tool to SW-/HW-Systems to get experience whether and how software reliability models can be applied to such systems. Model preselection, the selection criteria, aspects of the failure data used for model selection and calibration, scales (e.g., execution time vs. calendar time), the application of software reliability models to software-/hardware-systems, and the precision and usefulness of the results are discussed.

Safety and reliability analysis of complex systems will probably not be performed in a pure formal way. But formal and statistical techniques may contribute to enhance precision and reliability of the models. The traditional manual analyses are increasingly inappropriate. They are usually based on informal documents that describe the system. Considerable knowledge, system insight, and overview is necessary to consider many failure modes and dependencies between system components and their functionality at a time. Often, the behavior is too complicated to fully comprehend all possible failure consequences. Manual analysis is error-prone, costly and not necessarily complete. Formal and statistical techniques may be used to improve the reliability of the results.

Separation Logic for Small-Step Cminor

Andrew W. Appel[1,*] and Sandrine Blazy[2,*]

[1] Princeton University
[2] ENSIIE

Abstract. Cminor is a mid-level imperative programming language; there are proved-correct optimizing compilers from C to Cminor and from Cminor to machine language. We have redesigned Cminor so that it is suitable for Hoare Logic reasoning and we have designed a Separation Logic for Cminor. In this paper, we give a small-step semantics (instead of the big-step of the proved-correct compiler) that is motivated by the need to support future concurrent extensions. We detail a machine-checked proof of soundness of our Separation Logic. This is the first large-scale machine-checked proof of a Separation Logic w.r.t. a small-step semantics. The work presented in this paper has been carried out in the Coq proof assistant. It is a first step towards an environment in which concurrent Cminor programs can be verified using Separation Logic and also compiled by a proved-correct compiler with formal end-to-end correctness guarantees.

1 Introduction

The future of program verification is to connect machine-verified source programs to machine-verified compilers, and run the object code on machine-verified hardware. To connect the verifications end to end, the source language should be specified as a structural operational semantics (SOS) represented in a logical framework; the target architecture can also be specified that way. Proofs of source code can be done in the logical framework, or by other tools whose soundness is proved w.r.t. the SOS specification; these may be in safety proofs via type-checking, correctness proofs via Hoare Logic, or (in source languages designed for the purpose) correctness proofs by a more expressive proof theory. The compiler—if it is an optimizing compiler—will be a stack of phases, each with a well specified SOS of its own. There will be proofs of (partial) correctness of each compiler phase, or witness-driven recognizers for correct compilations, w.r.t. the SOS's that are inputs and outputs to the phases.

Machine-verified hardware/compiler/application stacks have been built before. Moore described a verified compiler for a "high-level assembly language" [13]. Leinenbach *et al.* [11] have built and proved a compiler for *C0*, a small C-like language, as part of a project to build machine-checked correctness proofs

* Appel supported in part by NSF Grants CCF-0540914 and CNS-0627650. This work was done, in part, while both authors were on sabbatical at INRIA.

K. Schneider and J. Brandt (Eds.): TPHOLs 2007, LNCS 4732, pp. 5–21, 2007.

of source programs, Hoare Logic, compiler, micro-kernel, and RISC processor. These are both simple one- or two-pass nonoptimizing compilers.

Leroy [12] has built and proved correct in Coq [1] a compiler called *CompCert* from a high-level intermediate language *Cminor* to assembly language for the Power PC architecture. This compiler has 4 intermediate languages, allowing optimizations at several natural levels of abstraction. Blazy *et al.* have built and proved correct a translator from a subset of C to Cminor [5]. Another compiler phase on top (not yet implemented) will then yield a proved-correct compiler from C to machine language. We should therefore reevaluate the conventional wisdom that an entire practical optimizing compiler cannot be proved correct.

A software system can have components written in different languages, and we would like end-to-end correctness proofs of the whole system. For this, we propose a new variant of Cminor as a machine-independent intermediate language to serve as a common denominator between high-level languages. Our new Cminor has a usable Hoare Logic, so that correctness proofs for some components can be done directly at the level of Cminor.

Cminor has a "calculus-like" view of local variables and procedures (*i.e.* local variables are bound in an environment), while Leinenbach's C0 has a "storage-allocation" view (*i.e.* local variables are stored in the stack frame). The calculus-like view will lead to easier reasoning about program transformations and easier use of Cminor as a target language, and fits naturally with a multi-pass optimizing compiler such as CompCert; the storage-allocation view suits the one-pass nonoptimizing C0 compiler and can accommodate in-line assembly code.

Cminor is a promising candidate as a common intermediate language for end-to-end correctness proofs. But we have many demands on our new variant of Cminor, only the first three of which are satisfied by Leroy's Cminor.

- Cminor has an operational semantics represented in a logical framework.
- There is a proved-correct compiler from Cminor to machine language.
- Cminor is usable as the high-level target language of a C compiler.
  - Our semantics is a *small-step* semantics, to support reasoning about input/output, concurrency, and nontermination.
  - Cminor is machine-independent over machines in the "standard model" (*i.e.* 32- or 64-bit single-address-space byte-addressable multiprocessors).
  - Cminor can be used as a mid-level target language of an ML compiler [8], or of an OO-language compiler, so that we can integrate correctness proofs of ML or OO programs with the proofs of their run-time systems and libraries.
  - As we show in this paper, Cminor supports an axiomatic Hoare Logic (in fact, Separation Logic), proved sound with respect to the small-step semantics, for reasoning about low-level (C-like) programs.
  - In future work, we plan to extend Cminor to be concurrent in the "standard model" of thread-based preemptive lock-synchronized weakly consistent shared-memory programming. The sequential soundness proofs we present here should be reusable in a concurrent setting, as we will explain.

Leroy's original Cminor had several Power-PC dependencies, is slightly clumsy to use as the target of an ML compiler, and is a bit clumsy to use in Hoare-style

reasoning. But most important, Leroy's semantics is a big-step semantics that can be used only to reason about terminating sequential programs. We have redesigned Cminor's syntax and semantics to achieve all of these goals. That part of the redesign to achieve target-machine portability was done by Leroy himself. Our redesign to ease its use as an ML back end and for Hoare Logic reasoning was fairly simple. Henceforth in this paper, Cminor will refer to the new version of the Cminor language.

The main contributions of this paper are a small-step semantics suitable for compilation and for Hoare Logic; and the first machine-checked proof of soundness of a sequential Hoare Logic (Separation Logic) w.r.t. a small-step semantics. Schirmer [17] has a machine-checked *big-step* Hoare-Logic soundness proof for a control flow much like ours, extended by Klein *et al.* [10] to a C-like memory model. Ni and Shao [14] have a machine-checked proof of soundness of a Hoare-like logic w.r.t. a small-step semantics, but for an assembly language and for much simpler assertions than ours.

2 Big-Step Expression Semantics

The C standard [2] describes a memory model that is byte- and word-addressable (yet portable to big-endian and little-endian machines) with a nontrivial semantics for uninitialized variables. Blazy and Leroy formalized this model [6] for the semantics of Cminor. In C, pointer arithmetic within any malloc'ed block is defined, but pointer arithmetic between different blocks is undefined; Cminor therefore has non-null pointer values comprising an abstract block-number and an int offset. A `NULL` pointer is represented by the integer value 0. Pointer arithmetic between blocks, and reading uninitialized variables, are undefined but not illegal: expressions in Cminor can evaluate to *undefined* (Vundef) without getting stuck.

Each memory load or store is to a non-null pointer value with a "chunk" descriptor *ch* specifying number of bytes, signed or unsigned, int or float. Storing as 32-bit-int then loading as 8-bit-signed-byte leads to an undefined value. Load and store operations on memory, $m \vdash v_1 \overset{ch}{\mapsto} v_2$ and $m' = m[v_1 \overset{ch}{:=} v_2]$, are partial functions that yield results only if reading (resp., writing) a chunk of type *ch* at address v_1 is legal. We write $m \vdash v_1 \overset{ch}{\mapsto} v$ to mean that the result of loading from memory m at address v_1 a chunk-type *ch* is the value v.

The *values* of Cminor are *undefined* (Vundef), integers, pointers, and floats. The int type is an abstract data-type of 32-bit modular arithmetic. The expressions of Cminor are literals, variables, primitive operators applied to arguments, and memory loads.

There are 33 primitive operation symbols *op*; two of these are for accessing global names and local stack-blocks, and the rest is for integer and floating-point arithmetic and comparisons. Among these operation symbols are casts. Cminor casts correspond to all portable C casts. Cminor has an infinite supply ident of variable and function identifiers *id*. As in C, there are two namespaces—each *id*