Olaf Owe
Martin Steffen
Jan Arne Telle (Eds.)

LNCS 6914

Fundamentals of Computation Theory

18th International Symposium, FCT 2011
Oslo, Norway, August 2011
Proceedings

Springer

Lecture Notes in Computer Science 6914

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

Olaf Owe Martin Steffen
Jan Arne Telle (Eds.)

Fundamentals of Computation Theory

18th International Symposium, FCT 2011
Oslo, Norway, August 22-25, 2011
Proceedings

Springer

Volume Editors

Olaf Owe
Martin Steffen
University of Oslo
Department of Informatics
Postboks 1080 Blindern, 0316 Oslo, Norway
E-mail: {olaf,msteffen}@ifi.uio.no

Jan Arne Telle
University of Bergen
Department of Informatics
Postboks 7800, 5020 Bergen, Norway
E-mail: jan.arne.telle@ii.uib.no

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-22952-7 e-ISBN 978-3-642-22953-4
DOI 10.1007/978-3-642-22953-4
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011933807

CR Subject Classification (1998): F.1, F.2, F.4, G.2

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the papers presented at FCT 2011: The 18th International Symposium on Fundamentals of Computer Theory held during August 22–25, 2011 in Oslo.

The Symposium on Fundamentals of Computation Theory was established in 1977 for researchers interested in all aspects of theoretical computer science, in particular in algorithms, complexity, and formal and logical methods. It is a biennial conference, which has previously been held in Poznań (1977), Wendisch-Rietz (1979), Szeged (1981), Borgholm (1983), Cottbus (1985), Kazan (1987), Szeged (1989), Gosen-Berlin (1991), Szeged (1993), Dresden (1995), Kraków (1997), Iaşi (1999), Riga (2001), Malmö (2003), Lübeck (2005), Budapest (2007), and Wrocław (2009).

The suggested topics of FCT 2011 included algorithms (algorithm design and optimization; combinatorics and analysis of algorithms; computational complexity; approximation, randomized, and heuristic methods; parallel and distributed computing; circuits and boolean functions; online algorithms; machine learning and artificial intelligence; computational geometry; and computational algebra), formal methods (algebraic and categorical methods; automata and formal languages; computability and nonstandard computing models; database theory; foundations of concurrency and distributed systems; logics and model checking; models of reactive, hybrid and stochastic systems; principles of programming languages; program analysis and transformation; specification, refinement and verification; security; and type systems) and emerging fields (ad hoc, dynamic, and evolving systems; algorithmic game theory; computational biology; foundations of cloud computing and ubiquitous systems; and quantum computation).

This year there were 78 reviewed submissions, of which 28 were accepted. The program included three invited talks, by Yuri Gurevich (Microsoft Research), Daniel Lokshtanov (UCSD), and José Meseguer (UIUC). This volume contains the accepted papers, abstracts from Yuri Gurevich and Daniel Lokshtanov, and a full paper on "The Rewriting Logic Semantics Project" by José Meseguer et al.

The symposium took place in the university informatics buildings, "Ole-Johan Dahls hus" and "Kristen Nygaards hus", and was one of the first scientific events to be held in the new building "Ole-Johan Dahls hus". The FCT event was part of the official opening of the new building and therefore augmented with a half-day program on Monday morning on the importance and future of object orientation, honoring the work of Ole-Johan Dahl and Kristen Nygaard. An additional invited speaker, Andrew P. Black (Portland State University), was invited for this session.

We would especially like to thank the members of the Program Committee for the evaluation of the submissions and their subreferees for excellent cooperation

in this work. We are grateful to the contributors to the conference, in particular to the invited speakers for their willingness to present interesting new developments.

Furthermore we thank the University of Oslo and the Department of Informatics for hosting the event, and we thank the local organization of the PMA group, in particular Johan Dovland, Cristian Prisacariu (Publicity Chair), Volker Stolz (Workshop Chair), Thi Mai Thoung Tran, and Ingrid Chieh Yu. Last but not least, we gratefully thank our sponsors: the Research Council of Norway, Cisco Systems Norway, DNV (Veritas) Norway, and the Department of Informatics.

June 2011

Olaf Owe
Martin Steffen
Jan Arne Telle

Organization

Program Committee

Wolfgang Ahrendt	Chalmers University of Technology, Sweden
David Coudert	INRIA Sophia Antipolis, France
Camil Demetrescu	Sapienza University of Rome, Italy
Johan Dovland	University of Oslo, Norway
Jiří Fiala	Charles University, Prague, Czech Republic
Martin Hofmann	LMU Munich, Germany
Thore Husfeldt	IT University of Copenhagen and Lund University, Sweden
Alexander Kurz	University of Leicester, UK
Andrzej Lingas	Lund University, Sweden
Olaf Owe	University of Oslo, Norway
Miguel Palomino	Universidad Complutense de Madrid, Spain
Yuri Rabinovich	Haifa University, Israel
Saket Saurabh	The Institute of Mathematical Sciences, Chennai, India
Kaisa Sere	Abo Akademi University, Turku, Finland
Martin Steffen	University of Oslo, Norway
Jan Arne Telle	University of Bergen, Norway
Tarmo Uustalu	Tallinn University of Technology, Estonia
Ryan Williams	IBM Almaden Research Center, San José, USA
Gerhard Woeginger	TU Eindhoven, The Netherlands
David R. Wood	The University of Melbourne, Australia
Wang Yi	Uppsala University, Sweden
Erika Ábrahám	RWTH Aachen, Germany
Peter Ölveczky	University of Oslo, Norway

Additional Reviewers

Antoniadis, Antonios	Bunde, David
Becchetti, Luca	Chatterjee, Krishnendu
Bentea, Lucian	Chen, Xin
Beringer, Lennart	Chlebikova, Janka
Birgisson, Arnar	Ciancia, Vincenzo
Bjorner, Nikolaj	Clarkson, Michael
Boronat, Artur	Cohen, Nathann
Bro Miltersen, Peter	Cortesi, Agostino

Corzilius, Florian
Crowston, Robert
D'Angelo, Gianlorenzo
Damaschke, Peter
Damiani, Ferruccio
Degerlund, Fredrik
Del Tedesco, Filippo
Dell, Holger
Delvenne, Jean-Charles
Dijk, Thomas C. Van
Din, Crystal Chang
Dorbec, Paul
Eklund, Tomas
Fernau, Henning
Fernández-Camacho, María Inés
Fleischer, Rudolf
Floderus, Peter
Fábregas, Ignacio
Gasieniec, Leszek
Gaspers, Serge
Gentilini, Raffaella
Giannopoulou, Archontia
Gopalan, Parikshit
Grabowski, Robert
Greenstreet, Mark
Hansen, Helle Hvid
Harrison, William
Hlineny, Petr
Hougardy, Stefan
Hähnle, Reiner
Jansen, Nils
Jervell, Herman Ruge
Jost, Steffen
Kabanets, Valentine
Kaufmann, Michael
Kazemeyni, Fatemeh
Kim, Eun Jung
Klasing, Ralf
Klin, Bartek
Kowaluk, Miroslaw
Kristiansen, Lars
Kulikov, Alexander
Laibinis, Linas
Laneve, Cosimo
Latte, Markus
Laud, Peeter
Leister, Wolfgang
Lepri, Daniela
Levcopoulos, Christos
Licata, Daniel R.
Lluch Lafuente, Alberto
Loup, Ulrich
M.S., Ramanujan
Marti-Oliet, Narciso
Marx, Dániel
Meister, Daniel
Mnich, Matthias
Monaco, Gianpiero
Nakata, Keiko
Nebel, Frank
Neovius, Mats
Niederreiter, Harald
Nilsson, Bengt
Norell, Ulf
Normann, Dag
Olsson, Roland
Palmigiano, Alessandra
Perdrix, Simon
Persson, Mia
Petre, Ion
Pettie, Seth
Piazza, Carla
Pitts, Andrew
Preoteasa, Viorel
Raman, Venkatesh
Ribichini, Andrea
Riedmiller, Martin
Rosa-Velardo, Fernando
Russo, Alejandro
Ryabko, Boris
Saha, Barna
Salomaa, Kai
Sampaio, Leonardo
Santos-Garcia, Gustavo
Sau, Ignasi
Schoepp, Ulrich
Segura, Clara
Shachnai, Hadas
Sledneu, Dzmitry
Soares, Ronan P.

Sorge, Manuel
Staton, Sam
Stojanovski, Toni
Tamm, Hellis
Tarasyuk, Anton
Thilikos, Dimitrios
Tulsiani, Madhur
Uehara, Ryuhei
Van Rooij, Johan M.M.
Verdejo, Alberto
Villanger, Yngve
Voge, Marie-Emilie
Watanabe, Osamu
Winterhof, Arne
Yan, Li
Ytrehus, Øyvind
Yu, Ingrid
Zantema, Hans

Table of Contents

The Rewriting Logic Semantics Project: A Progress Report

José Meseguer and Grigore Roşu

Department of Computer Science,
University of Illinois at Urbana-Champaign,
{meseguer,grosu}@illinois.edu

Abstract. Rewriting logic is an executable logical framework well suited for the semantic definition of languages. Any such framework has to be judged by its effectiveness to bridge the existing gap between language definitions on the one hand, and language implementations and language analysis tools on the other. We give a progress report on how researchers in the rewriting logic semantics project are narrowing the gap between theory and practice in areas such as: modular semantic definitions of languages; scalability to real languages; support for real time; semantics of software and hardware modeling languages; and semantics-based analysis tools such as static analyzers, model checkers, and program provers.

1 Introduction

The disconnect between theory and practice is one of the worse evils in computer science. Theory disconnected from practice becomes irrelevant; and practice without theory becomes brute-force, costly and ad-hoc engineering. One of the current challenges in formal approaches to language semantics is precisely how to effectively bridge the gap between theory and practice. There are two distinct dimensions to this gap:

(1) Given a language $\mathcal{L}$, there is often a substantial gap between: (i) a formal semantics for $\mathcal{L}$; (ii) an implementation of $\mathcal{L}$; and (iii) analysis tools for $\mathcal{L}$, including static, dynamic, and deductive tools.
(2) Even if a formal semantics exists for a programming language $\mathcal{L}$, there may not be any formal semantics available at the higher level of software designs and models, or at the lower level of hardware.

Regarding (1), a semantics of $\mathcal{L}$ may just be a "paper semantics," such as some SOS rules on a piece of paper; or it may be a "toy semantics," not for $\mathcal{L}$ itself, but for a greatly simplified sublanguage. Furthermore, the way a compiler for $\mathcal{L}$ is written may have no connection whatever with a formal semantics for $\mathcal{L}$, so that different compilers provide different language behaviors. To make things worse, program analysis tools for $\mathcal{L}$, including tools that supposedly provide some formal analysis, may not be systematically based on a formal semantics either, so that the confidence one can place of the answers from such tools is greatly

O. Owe, M. Steffen and J.A. Telle (Eds.): FCT 2011, LNCS 6914, pp. 1–37, 2011.

diminished. Regarding (2), one big problem is that software modeling notations often lack a formal semantics. A related problem is that this lack of semantics manifests itself as a lack of *analytic power*, that is, as an incapacity to uncover expensive design errors which could have been caught by formal analysis.

We, together with many other colleagues all over the world, have been working for years on the *rewriting logic semantics project* (see [77, 76, 112] for some overview papers at different stages of the project). The goal of this project is to substantially narrow the gap between theory and practice in language specifications, implementations and tools, in both of the above dimensions (1)–(2). In this sense, rewriting logic semantics is a *wide-spectrum framework*, where:

1. The formal semantics of a language $\mathcal{L}$ is used as the *basis* on which both language implementations and language analysis tools are built.
2. The same semantics-based approach is used not just for programming languages, but also for software and hardware modeling languages.

Any attempt to bridge theory and practice cannot be judged by theoretical considerations alone. One has to evaluate the practical effectiveness of the approach in answering questions such as the following:

- *Executability.* Is the semantics executable? How efficiently so? Can semantic definitions be tested to validate their agreement with an informal semantics?
- *Range of Applicability.* Can it be applied to programming languages and to software and hardware modeling languages? Can it naturally support nontrivial features such as concurrency and real time?
- *Scalability.* Can it be used in practice to give full definitions of real languages like Java or C? And of real software and hardware modeling languages?
- *Integrability.* How well can the semantics be integrated with language implementations and language analysis tools? Can it really be used as the *basis* on which such implementations and analysis tools are built?

This paper is a progress report on the efforts by various researchers in the rewriting logic semantics project to positively answer these questions. After summarizing some related work below, we give an overview of rewriting logic semantics in Section 2. Subsequent sections then describe in more detail: (i) modularity of definitions and the support for highly modular definitions provided by the $\mathbb{K}$ framework (Section 3); (ii) semantics of programming languages (Section 4); semantics of real-time language (Section 5); (iv) semantics of software modeling languages (Section 6); (v) semantics of hardware description languages (Section 7); (vi) abstract semantics and static analysis (Section 8); (vii) model checking verification (Section 9); and (viii) deductive verification (Section 10). We finish with some concluding remarks in Section 11.

1.1 Related Work

There is much related work on frameworks for defining programming languages. Without trying to be exhaustive, we mention some of them and point out some relationships to rewriting logic semantics (RLS).

Structural Operational Semantics (SOS). Several variants of structural operational semantics have been proposed. We refer to [112] for an in-depth comparison between SOS and RLS. A key point made in [112], and also made in Section 2.5, is that RLS is a framework supporting many different definitional styles. In particular, it can naturally and faithfully express many diffent SOS styles such as: small-step SOS [99], big-step SOS [56], MSOS [87], reduction semantics [129], continuation-based semantics [43], and the CHAM [12].

Algebraic denotational semantics. This approach, (see [125, 49, 26, 85] for early papers and [47,118] for two more recent books), is the special case of RLS where the rewrite theory $\mathcal{R}_{\mathcal{L}}$ defining a language $\mathcal{L}$ is an equational theory. Its main limitation is that it is well suited for giving semantics to *deterministic* languages, but not well suited for concurrent language definitions.

Higher-order approaches. The most classic higher-order approach is *denotational semantics* [109, 110, 108, 86]. Denotational semantics has some similarities with its first-order algebraic cousin mentioned above, since both are based on semantic equations and both are best suited for deterministic languages. Higher-order functional languages or higher-order theorem provers can be used to give an executable semantics to programming languages, including the use of Scheme in [45], the use of ML in [98], and the use of Common LISP within the ACL2 prover in [61]. There is also a body of work on using monads [81, 124, 65] to implement language interpreters in higher-order functional languages; the monadic approach has better modularity characteristics than standard SOS. Some higher-order approaches are based on the use of higher-order abstract syntax (HOAS) [97, 52] and higher-order logical frameworks, such as LF [52] or λ-Prolog [88], to encode programming languages as formal logical systems; for a good example of recent work in this direction see [78] and references there.

Logic-programming-based approaches. Going back to the Centaur project [22, 35], logic programming has been used as a framework for SOS language definitions. Note that λ-Prolog [88] belongs both in this category and in the higher-order one. For a recent textbook giving logic-programming-based language definitions, see [113].

Abstract state machines. Abstract State Machine (ASM) [50] can encode any computation and have a rigorous semantics, so any programming language can be defined as an ASM and thus implicitly be given a semantics. Both big- and small-step ASM semantics have been investigated. The semantics of various programming languages, including Java [114], has been given using ASMs.

Other RLS work. RLS is a collective international project. There is by now a substantial body of work demonstrating the usefulness of this approach, e.g., [23, 120, 117, 115, 72, 119, 31, 104, 122, 42, 40, 55, 25, 73, 77, 30, 28, 41, 34, 106, 1, 116, 36, 107, 58, 54, 46, 39, 5], and we describe some even more recent advances in this paper. A first snapshot of the RLS project was given in [77], a second in [76], and a third in [112], with this paper as the fourth snapshot.