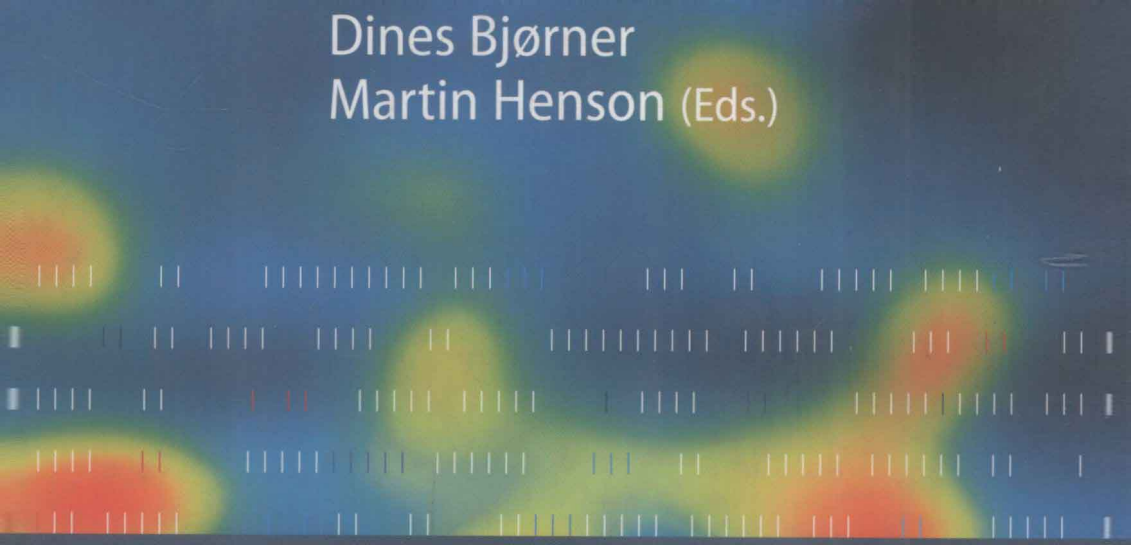
Dines Bjørner
Martin Henson (Eds.)



Logics of Specification Languages

Springer

Dines Bjørner · Martin C. Henson
Editors

Logics of Specification Languages

Springer

Prof. Emeritus, Dr. Dines Bjørner
Informatics and Mathematical Modelling
Technical University of Denmark
2800 Kgs. Lyngby
Denmark
bjorner@gmail.com

Prof. Martin C. Henson
University of Essex
Department of Computer Science
Wivenhoe Park
CO4 3SQ Colchester
United Kingdom
hensm@essex.ac.uk

ISBN 978-3-540-74106-0 e-ISBN 978-3-540-74107-7

DOI 10.1007/978-3-540-74107-7

ACM Computing Classification (1998): F.4, F.3, D.1, D.2, D.3

Library of Congress Control Number: 2007936401

Monographs in Theoretical Computer Science. An EATCS Series. ISSN 1431-2654

Cover Design: KünkelLopka, Heidelberg

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Monographs in Theoretical Computer Science

An EATCS Series

For this we need resort to the proof system of the specification language — as well as to other means. We consider in this prelude three such means.

Verification

Verification, in general terms, is a wide and inclusive term covering all approaches which have the aim of establishing that a system meets certain properties. Even a simple *test case* demonstrates a, perhaps limited, fact: that in *this* case (though maybe no others) a given system achieves (or does not) a desirable outcome.

More specifically and usually, we use the term *verification* for more elaborate and systematic mathematical techniques for establishing that systems possess certain properties. Here, the *system* might be a more-or-less abstract description (a specification) or a concrete realisation in hardware or software. The *properties* may be specific emergent properties of abstract specifications; they include general statements of, say, *liveness*, *safety* and/or *termination*; and they cover the *correctness* of realisations or implementations of given system specifications. In all the cases of interest to us, the system description and the properties to be determined will be couched in a precise formal mathematical language. As a consequence, the results of such a verification will be correspondingly precise and formal.

There are three forms of formal verification that are relevant to the material covered in this book and that are, therefore, worth describing in just a little more detail.

Inferential Verification

This approach is often simply referred to as *verification* despite the fact that other approaches, such as model checking, are also such methods. Here, we have at our disposal logical principles, a logic or proof system, which correctly captures the framework within which the system is described. This framework might be a programming or specification language with a semantics which lays down, normatively, its meaning. The logical principles will (at the very least) be *sound* with respect to that semantics; thus ensuring that any conclusions drawn will be correct judgements of the language in question.

The logical principles, or fully-fledged logic, will provide means that are appropriate for reasoning about the techniques and mechanisms that are available in the language of description. For example, many frameworks provide a means for describing recursive systems, and appropriate induction principles are then available for reasoning about such systems.

Inference-based methods of verification allow us to make and support general claims about a system. These may demonstrate that an implementation is *always* guaranteed to meet its specification; that it *always* possesses certain characteristic properties (for example, that it is *deadlock-free* or maybe that it

terminates); or that an abstract specification will always possess certain implicit properties (which will, in turn, be inherited properties of *any* (correct) implementation).

Model Checking

This approach to verification (see, for example, [6]) aims to automatically establish (or provide a counterexample for) a property by direct inspection of a model of the system in question. The model may be represented (explicitly or implicitly) by a directed graph whose nodes are states and whose edges are legitimate state transitions; properties may be expressed in some form of temporal logic.

Two key issues are *finiteness* and the potential *combinatorial explosion* of the state space. Many techniques have been developed to minimise the search. In many cases it is not necessary to build the state graph but simply to represent it symbolically, for example by propositional formulae, and then, using techniques such as SAT-solvers, to mimic the graph search. Partial order reductions, which remove redundancies (in explicit graphs) arising from independent interleavings of concurrent events can also be employed to significantly reduce the size of the search space. It is also possible to simplify the system, through abstraction, and to investigate the simpler model as a surrogate for the original system. This, of course, requires that the original and abstracted systems are related (by refinement) and that the abstracted system is at least *sound* (if not *complete*) with respect to the original: that properties true of the abstracted system are also true of the original, even if the abstracted system does not capture *all* properties of the original.

Model checking has been a spectacularly successful technology by any measure; the model checker SPIN [23], for example, detected several crucial errors in the controller for a spacecraft [21]. Other important model checkers are SMV [31] and FDR, based on the standard *failures-divergencies* model of CSP [42].

Formal Testing

Dijkstra, in his ACM Turing Lecture in 1972, famously said: "... *program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence*" [9]. A correct contrast between informal testing (which might demonstrate a flaw in a system) and a formal verification (which might make a general correctness claim) was established by this remark. More recently, however, it has become clear that there is something to be gained by combining variations on the general theme of testing with formal specifications and verifications. Indeed, the failure of a *formal* test is a *counterexample*, which is as standard a mathematical result as could be wished for (and potentially as valuable too); the problem is that when testing

without a theoretical basis (informal testing), it is often simply unclear what conclusion can and should be drawn from such a methodology.

A portfolio approach, in which a variety of verification methods are used, brings benefits. In the case of *formal* testing, there is an interplay between test (creation, application and analysis) and system specification: a formal description of a system is an excellent basis for the generation (possibly automatically) of test cases which, themselves, have precise properties regarding coverage, correctness and so on. In addition, the creation of adequate test suites is expensive and time-consuming, not to say repetitious if requirements and specifications evolve; exploiting the precision implicit in formal specification to aid the creation of test suites is a major benefit of formal testing technologies.

1.3 Integration of Specification Languages

Domains, requirements or software being described, prescribed or designed, respectively, usually possess properties that cannot be suitably specified in one language only. Typically a variety, a composition, a "mix" of specification notations need be deployed. In addition to, for example, either of ASM, B, CafeOBJ, CASL, RAISE/RSL, VDM or Z, the specifier may resort to additionally using one or more (sometimes diagrammatic) notations such as Petri nets [27, 35, 37–39], message sequence charts [24–26], live sequence charts [7, 19, 28], statecharts [15–18, 20], and/or some textual notations such as temporal logics (Duration Calculus, TLA+, or LTL — for linear temporal logic [10, 29, 30, 34, 36]).

Using two or more notations, that is, two or more semantics, requires their integration: that an identifier a in one specification (expressed in one language) and "the same" identifier (a) in another specification (in another language) can be semantically related (i.e., that there is a 'satisfaction relation').

This issue of integrating formal tools and techniques is currently receiving high attention as witnessed by many papers and a series of conferences: [1, 3, 4, 13, 41]. The present book will basically not cover integration.[1]

2 Structure of Book

The book is structured as follows: In the main part, Part II, we introduce, in alphabetic order, nine chapters on ASM, event-B, CafeOBJ, CASL, DC, RAISE, TLA$^+$, VDM and Z. Each chapter is freestanding: It has its own list of references and its own pair of symbol and concept indexes. Part III introduces just one chapter, Review, in which eight "originators" of respective specification languages will comment briefly on the chapter on "that language".

[1] TLA$^+$ can be said to be an integration of a temporal logic of actions, TLA, with set-theoretical specification. The RAISE specification language has been "integrated" with both Duration Calculus and concrete timing.

3 Acknowledgements

Many different kinds of institutions and people must be gratefully acknowledged.

CoLogNET: Dines Bjørner thanks the 5th EU/IST Framework Programme (`http://www.cordis.lu/fp5/home.html`) of the European Commission, Contract Reference IST-2001-33123: CoLogNET: Network of Excellence in Computational Logic: `http://www.eurice.de/colognet` for support.

CAI: Dines Bjørner thanks the editorial board of the Slovak Academy Journal for giving us the opportunity to publish the papers mentioned on Pages 4–5.

Stara Lesna: We both thank Dr. Martin Pěnička of the Czech Technical University in Prague and Prof. Branislav Rovan and Dr. Dusan Guller of Comenius University in Bratislava, Slovakia for their support in organising the Summer School mentioned on Pages 5–6.

Book Preparation: We both thank all the contributing authors for their willingness to provide their contributions and their endurance also during the latter parts of the editing phase.

Springer: We both thank the editorial board of the EATCS Monographs in Theoretical Computer Science Series and the Springer editor, Ronan Nugent, for their support in furthering the aims of this book.

Our Universities: Last, but not least, we gratefully acknowledge our universities for providing the basis for this work: the University of Essex, UK and the Technical University of Denmark (DTU).

Martin Henson
University of Essex
Colchester, UK
April 4, 2007

Dines Bjørner
Technical University of Denmark
Kgs. Lyngby, Denmark
April 4, 2007

References

1. K. Araki, A. Galloway, K. Taguchi, editors. *IFM 1999: Integrated Formal Methods*, volume 1945 of *Lecture Notes in Computer Science*, York, UK, June 1999. Springer. Proceedings of 1st Intl. Conf. on IFM.
2. Edited by D. Bjørner, M.C. Henson: *Logics of Specification Languages* (Springer, 2007)

3. E.A. Boiten, J. Derrick, G. Smith, editors. *IFM 2004: Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, London, UK, April 4–7 2004. Springer. Proceedings of 4th Intl. Conf. on IFM.
4. M.J. Butler, L. Petre, K. Sere, editors. *IFM 2002: Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, Turku, Finland, May 15–18 2002. Springer. Proceedings of 3rd Intl. Conf. on IFM.
5. D. Cansell, D. Méry. *The event-B Modelling Method: Concepts and Case Studies*, pages 33–138. Springer, 2007. See [2].
6. E.M. Clarke, O. Grumberg, D.A. Peled: *Model Checking* (MIT Press, 2000)
7. W. Damm, D. Harel: *LSCs: Breathing Life into Message Sequence Charts.* Formal Methods in System Design **19** (2001) pages 45–80
8. R. Diaconescu. *A Methodological Guide to CafeOBJ Logic*, pages 139–218. Springer, 2007. See [2].
9. E.W. Dijkstra: *The Humble Programmer.* Communications of the ACM **15**, 10 (1972) pages 859–866
10. B. Dutertre: Complete Proof System for First-Order Interval Temporal Logic. In: *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science* (IEEE CS, 1995) pages 36–43
11. J.S. Fitzgerald. *The Typed Logic of Partial Functions and the Vienna Development Method*, pages 427–461. Springer, 2007. See [2].
12. C. George, A.E. Haxthausen. *The Logic of the RAISE Specification Language*, pages 325–375. Springer, 2007. See [2].
13. W. Grieskamp, T. Santen, B. Stoddart, editors. *IFM 2000: Integrated Formal Methods*, volume of *Lecture Notes in Computer Science*, Schloss Dagstuhl, Germany, November 1–3 2000. Springer. Proceedings of 2nd Intl. Conf. on IFM.
14. M.R. Hansen. *Duration Calculus*, pages 277–324. Springer, 2007. See [2].
15. D. Harel: *Statecharts: A Visual Formalism for Complex Systems.* Science of Computer Programming **8**, 3 (1987) pages 231–274
16. D. Harel: *On Visual Formalisms.* Communications of the ACM **33**, 5 (1988)
17. D. Harel, E. Gery: *Executable Object Modeling with Statecharts.* IEEE Computer **30**, 7 (1997) pages 31–42
18. D. Harel, H. Lachover, A. Naamad et al.: *STATEMATE: A Working Environment for the Development of Complex Reactive Systems.* Software Engineering **16**, 4 (1990) pages 403–414
19. D. Harel, R. Marelly: *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine* (Springer, 2003)
20. D. Harel, A. Naamad: *The STATEMATE Semantics of Statecharts.* ACM Transactions on Software Engineering and Methodology (TOSEM) **5**, 4 (1996) pages 293–333
21. K. Havelund, M.R. Lowry, J. Penix: *Formal Analysis of a Space Craft Controller Using SPIN.* Software Engineering **27**, 8 (2001) pages 1000–9999
22. M.C. Henson, M. Deutsch, S. Reeves. *Z Logic and Its Applications*, pages 463–565. Springer, 2007. See [2].
23. G.J. Holzmann: *The SPIN Model Checker: Primer and Reference Manual* (Addison-Wesley Professional, 2003)
24. ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992.
25. ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1996.
26. ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1999.

27. K. Jensen: *Coloured Petri Nets*, vol 1: Basic Concepts (234 pages + xii), Vol. 2: Analysis Methods (174 pages + x), Vol. 3: Practical Use (265 pages + xi) of *EATCS Monographs in Theoretical Computer Science* (Springer–Verlag, Heidelberg 1985, revised and corrected second version: 1997)
28. J. Klose, H. Wittke: An Automata Based Interpretation of Live Sequence Charts. In: *TACAS 2001*, ed by T. Margaria, W. Yi (Springer-Verlag, 2001) pages 512–527
29. Z. Manna, A. Pnueli: *The Temporal Logic of Reactive Systems: Specifications* (Addison-Wesley, 1991)
30. Z. Manna, A. Pnueli: *The Temporal Logic of Reactive Systems: Safety* (Addison-Wesley, 1995)
31. K. McMillan: *Symbolic Model Checking* (Kluwer, Amsterdam 1993)
32. S. Merz. *The Specification Language TLA$^+$*, pages 377–426. Springer, 2007. See [2].
33. T. Mossakowski, A.E. Haxthausen, D. Sannella, A. Tarlecki. *CASL – The Common Algebraic Specification Language*, pages 219–276. Springer, 2007. See [2].
34. B.C. Moszkowski: *Executing Temporal Logic Programs* (Cambridge University Press, UK 1986)
35. C.A. Petri: *Kommunikation mit Automaten* (Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962)
36. A. Pnueli: The Temporal Logic of Programs. In: *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science* (IEEE CS, 1977) pp 46–57
37. W. Reisig: *Petri Nets: An Introduction*, vol 4 of *EATCS Monographs in Theoretical Computer Science* (Springer, 1985)
38. W. Reisig: *A Primer in Petri Net Design* (Springer, 1992)
39. W. Reisig: *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets* (Springer, 1998)
40. W. Reisig. *Abstract State Machines for the Classroom*, pages 1–32. Springer, 2007. See [2].
41. J.M. Romijn, G.P. Smith, J.C. van de Pol, editors. *IFM 2005: Integrated Formal Methods*, volume 3771 of *Lecture Notes in Computer Science*, Eindhoven, The Netherlands, December 2005. Springer. Proceedings of 5th Intl. Conf. on IFM.
42. A.W. Roscoe: *The Theory and Practice of Concurrency* (Prentice Hall, 1999)

List of Contributors

Jean-Raymond Abrial
Department of Computer Science
Swiss Fed. Univ. of Technology
Haldeneggsteig 4/Weinbergstrasse
CH-8092 Zürich
Switzerland
jabrial@inf.ethz.ch

Dines Bjørner
Informatics
and Mathematical Modelling
Technical University of Denmark
DK-2800 Kgs. Lyngby
Denmark
bjorner@gmail.com

Dominique Cansell
LORIA
Campus scientifique, BP 239
F-54506 Vandœuvre-lès-Nancy
France
cansell@loria.fr

Moshe Deutsch
Hagefen 45
Moshav Liman
22820 Israel
Moshe.Deutsch@Alvarion.com

Răzvan Diaconescu
Inst. of Math. "Simion Stoilow"
PO Box 1-764
Bucharest 014700
Romania
Razvan.Diaconescu@imar.ro

John Fitzgerald
Centre for Software Reliability
School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK
John.Fitzgerald@ncl.ac.uk

Kokichi Futatsugi
Japan Adv. Inst. of Sci. & Techn.
1-1 Asahidai, Nomi,
Ishikawa, 923-1292 Japan
kokichi@jaist.ac.jp

Chris George
United Nations University
Intl. Inst. for Software Technology
Casa Silva Mendes
Est. do Engenheiro Trigo No. 4
P.O. Box 3058
Macau, China
cwg@iist.unu.edu

Yuri Gurevich
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
gurevich@microsoft.com

Michael R. Hansen
Informatics
and Mathematical Modelling
Technical University of Denmark
DK-2800 Kgs. Lyngby
Denmark
mrh@imm.dtu.dk

Klaus Havelund
Lab. for Reliable Software
Jet Propulsion Laboratory (JPL)
4800 Oak Grove Drive
M/S 301-285
Pasadena, CA 91109
USA
havelund@gmail.com

Anne E. Haxthausen
Informatics
and Mathematical Modelling,
Technical University of Denmark
DK-2800 Kgs. Lyngby
Denmark
ah@imm.dtu.dk

Martin C. Henson
Department of Computer Science
University of Essex
Wivenhoe Park
Colchester
Essex CO4 3SQ
UK
hensm@essex.ac.uk

Cliff Jones
School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU
UK
cliff.jones@ncl.ac.uk

Leslie Lamport
Microsoft Corporation
1065 La Avenida
Mountain View, CA 94043
USA
lamport@microsoft.com

Dominique Méry
LORIA
Campus scientifique, BP 239
F-54506 Vandœuvre-lès-Nancy
France
Dominique.Mery@loria.fr

Stephan Merz
INRIA Lorraine
Equipe MOSEL
Bâtiment B
615, rue du Jardin Botanique
F-54602 Villers-lès-Nancy
France
Stephan.Merz@loria.fr

Till Mossakovski
DFKI Lab Bremen
Robert-Hooke-Str. 5
DE-28359 Bremen
Germany
till@informatik.uni-bremen.de

Peter D. Mosses
Dept of Computer Science
Swansea University
Singleton Park
Swansea SA2 8PP
UK
P.D.Mosses@swansea.ac.uk

Steve Reeves
Computer Science Department
Computing
& Mathematical Sciences
University of Waikato
Private Bag 3105
Hamilton
New Zealand
stever@cs.waikato.ac.nz

Wolfgang Reisig
Institut für Informatik
Math.-Nat. Fakultät II
Humboldt-Universität zu Berlin
Unter den Linden 6, DE 10099 Berlin
Germany
reisig@informatik.hu-berlin.de

Donald Sannella
LFCS, School of Informatics
University of Edinburgh
King's Buildings
Mayfield Road
Edinburgh EH9 3JZ
UK
dts@inf.ed.ac.uk

Andrzej Tarlecki
Institute of Informatics
Warsaw University
ul. Banacha 2
PL 02-097 Warsaw
Poland
tarlecki@mimuw.edu.pl

Zhou Chaochen
Institute of Software
Chinese Academy of Sciences
P.O. Box 8718
100080 Beijing
China
zcc@ios.ac.cn

Contents