

## References

1. S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. In *Proc. 17th ACM Sympos. Principles of Database Systems (PODS 1998)*, pp. 179–187. ACM Press, 1998.  
Relational transducers mapping sequences of input relations to sequences of output relations are proposed for high-level declarative specifications of business models. See [404] for a related class of ASM-transducers.
2. U. Abraham. *Models for Concurrency*. Gordon and Breach, 1999.
3. U. Abraham. Bakery algorithms. Manuscript of 41 pages from University of Beer Sheva, Nov 19, 2001.
4. S. Abramsky. Semantics of interaction. In A. Pitts and P. Dybjer (eds.), *Semantics and Logics of Computation*, pp. 1–31. Cambridge University Press, Cambridge, 1997.
5. J.-R. Abrial. *The B-Book*. Cambridge University Press, Cambridge, 1996.
6. J.-R. Abrial. Extending B without changing it (for developing distributed systems). In H. Habrias (ed.), *Proc. 1st Conf. on the B Method*, pp. 169–190. IRIN Institut de recherche en informatique de Nantes, 1996.
7. J.-R. Abrial, E. Börger, and H. Langmaack. *Methods for Semantics and Specification*, Vol. 117. Dagstuhl Seminar No. 9523, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, May 1995.
8. J.-R. Abrial, E. Börger, and H. Langmaack (eds.). *Formal Methods for Industrial Applications. Specifying and Programming the Steam Boiler Control*, Lecture Notes in Computer Science, Vol. 1165. Springer-Verlag, 1996.  
Contains the problem description for the steam boiler control competition [7] and 22 proposed solutions obtained using the major known formal methods, with text and complete documentation on the accompanying CD.
9. J.-R. Abrial, E. Börger, and H. Langmaack. The steam boiler case study: Competition of formal program specification and development methods. In J.-R. Abrial, E. Börger, and H. Langmaack (eds.), *Formal Methods for Industrial Applications. Specifying and Programming the Steam-Boiler Control*, Lecture Notes in Computer Science, Vol. 1165, pp. 1–12. Springer-Verlag, 1996.  
Motivation of the steam-boiler control competition [7] and short characterization of the 22 problem solutions appearing in the book.
10. J.-R. Abrial and L. Mussat. Specification and design of a transmission protocol by successive refinements using B. In M. Broy and B. Schieder (eds.), *Mathematical Methods in Program Development*. Springer-Verlag, 1996.
11. J.-R. Abrial and L. Mussat. Introducing dynamic constraints in B. In D. Bert (ed.), *B'98: Recent Advances in the Development and Use of the B Method*, Lecture Notes in Computer Science, Vol. 1393, pp. 82–128. Springer-Verlag, 1998.

12. M. Allemand, C. Attiogbé, and H. Habrias (eds.). *Comparing Systems Specification Techniques.*, ISBN 2-906082-29-5. IRIN Nantes, March 1998.
13. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
14. R. Alur and T. Henzinger. Real-time system = discrete system + clock variables. *Software Tools for Technology Transfer*, 1:86–109, 1997.
15. M. Anlauff. XASM – an extensible, component-based Abstract State Machines language. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 69–90. Springer-Verlag, 2000.  
The XASM (Extensible ASM) language for producing executable ASMs is presented. A development environment for XASM systems is described. (XASM was formerly known as Aslan.) Also appears in TIK-Report No. 87, pp. 1–21, ETH Zürich, March 2000. See [17].
16. M. Anlauff, S. Chakraborty, P. Kutter, A. Pierantonio, and L. Thiele. Generating an Action Notation environment from Montages descriptions. *Software Tools and Technology Transfer*, 3:431–455, 2001.  
Montages [19] are used to provide executable semantics for Action Notation. A preliminary version was published by M. Anlauff, P. Kutter, A. Pierantonio and L. Thiele under the title “Generating an Action Notation Environment from Montages Descriptions” in Proc. 2nd Int. Workshop on Action Semantics (AS’99), ed. by P. Mosses and D. Watt, University of Aarhus, Department of Computer Science, BRICS Notes Series NS-99-3, March 1999, pp. 1–42.
17. M. Anlauff and P. Kutter. Xasm Open Source. Web pages at <http://www.xasm.org/>, 2001.  
The extensible ASM project [15] is Open Source.
18. M. Anlauff, P. Kutter, and A. Pierantonio. Montages/Gem-Mex: a meta visual programming generator. TIK-Report 35, ETH Zürich, Switzerland, 1998.  
An introduction to Montages [311] and GEM-MEX, the development tool for creating Montages (using a graphical editor) and generating language interpreters from them, supporting also debugging and animation features. A description of their use and some small examples can be found in *Formal Aspects of and Development Environments for Montages* by the same authors, published in M. Sellink (ed.): 2nd Int. Workshop on the Theory and Practice of Algebraic Specifications, Springer Workshops in Computing 1997. Another description is in *Tool Support for Language Design and Prototyping with Montages* published by the same authors in Proc. of Compiler Construction (CC’99), Springer Lecture Notes in Computer Science, Vol. 1575, pp. 296–300, Springer-Verlag 1999. Another illustration is in [20].
19. M. Anlauff, P. Kutter, and A. Pierantonio. Enhanced control flow graphs in Montages. In D. Bjoerner and M. Broy (eds.), *Proc. Perspectives of System Informatics (PSI’99)*, Lecture Notes in Computer Science, Vol. 1755, pp. 40–53. Springer-Verlag, 1999.  
A refined definition of Montages, based on the notion of finite state machines, triggered by the use of Montages for defining the static semantics of Java in [421] which showed some shortcomings of the original formulation in [311].
20. M. Anlauff, P. Kutter, A. Pierantonio, and A. Sünbül. Using domain-specific languages for the realization of component composition. In T. Maibaum

- (ed.), *Fundamental Approaches to Software Engineering (FASE 2000)*, Lecture Notes in Computer Science, Vol. 1783, pp. 112–126, 2000.
- An illustration of how to apply Montages [311, 19] and of its tool environment Gem-Mex [18] for the implementation of component interaction.
21. L. Araujo. Correctness proof of a distributed implementation of Prolog by means of Abstract State Machines. *J. Universal Computer Science*, 3(5):416–422, 1997.
 

Building upon [132], a specification and a proof of correctness for the Prolog Distributed Processor (PDP), a WAM extension for parallel execution of Prolog on distributed memory, are provided. A preliminary version appeared in 1996 under the title *Correctness Proof of a Parallel Implementation of Prolog by Means of Evolving Algebras* as Technical Report DIA 21-96 of Dpto. Informática y Automática, Universidad Complutense de Madrid.
  22. U. Assmann, A. Heberle, W. Löwe, A. Ludwig, and R. Neumann. Language concepts and design patterns. Manuscript, 1999.
 

ASMs are used to define the semantics of patterns and for correctness proofs for workarounds.
  23. V. Awahad and C. Wallace. A unified formal specification and analysis of the new Java memory models. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 166–185. Springer-Verlag, 2003.
 

See [421].
  24. R. J. R. Back. On correct refinement of programs. *J. Computer and System Sciences*, 23(1):49–68, 1979.
  25. R. J. R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.
  26. M. Barnett, E. Börger, Y. Gurevich, W. Schulte, and M. Veanes. Using Abstract State Machines at Microsoft: A case study. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 367–380. Springer-Verlag, 2000.
 

A description of a reverse-engineering case study, modeling a command-line debugger of a stack-based run-time environment at three levels of abstraction. For a powerpoint slide show see **Debugger** ( $\rightsquigarrow$  CD).
  27. M. Barnett, C. Campbell, W. Schulte, and M. Veanes. Specification, simulation and testing of COM components using Abstract State Machines. In R. Moreno-Díaz and A. Quesada-Arencibia (eds.), *Formal Methods and Tools for Computer Science (Local Proceedings of Eurocast 2001)*, pp. 266–270, Canary Islands, Spain, February 2001. Universidad de Las Palmas de Gran Canaria.
 

A description of the use of AsmL to specify, simulate, and test the interfaces of Microsoft COM components.
  28. M. Barnett, L. Nachmanson, and W. Schulte. Conformance checking of components against their non-deterministic specifications. Technical Report MSR-TR-2001-56, Microsoft Research, Redmond, Washington, June 2001.
 

A method for testing a Microsoft COM (Component Object Model) component against a (possibly non-deterministic) ASM specification is presented. See [31].

29. M. Barnett and W. Schulte. Spying on components: A runtime verification technique. In G. T. Leavens, M. Sitaraman, and D. Giannakopoulou (eds.), *Workshop on Specification and Verification of Component-Based Systems*, pp. 7–13. Technical Report TR 01-09a, Iowa State University, 2001.  
A predecessor of [31].
30. M. Barnett and W. Schulte. The ABCs of specification: AsmL, behavior, and components. *Informatica*, 25(4):517–526, 2002.  
See [31].
31. M. Barnett and W. Schulte. Contracts, components and their runtime verification on the .NET platform. *J. Systems and Software*, Special Issue on Component-Based Software Engineering, 2002, to appear.  
Continuing [28, 29, 30] AsmL is proposed to implement behavioral interface specifications, including component interaction, on the .NET platform.
32. A. Bartoloni et al. A hardware implementation of the APE100 architecture. *Int. J. Mod. Phys.*, C(4):969, 1993.  
The architecture which has been chosen by Börger for Del Castillo’s Tesi di Laurea to try out ASMs for modeling real-world architectures, in the context of a reengineering project for this machine which had been launched by a group of physicists in Pisa and Rome; see [97],[102].
33. A. Bartoloni et al. The software of the APE100 architecture. *Int. J. Mod. Phys.*, C(4):955, 1993.  
See comment to [32].
34. D. Beauquier and A. Slissenko. The railroad crossing problem: Towards semantics of timed algorithms and their model-checking in high-level languages. In M. Bidoit and M. Dauchet (eds.), *TAPSOFT’97: Theory and Practice of Software Development, 7th Int. Joint Conf. CAAP/FASE*, Lecture Notes in Computer Science, Vol. 1214, pp. 201–212. Springer-Verlag, 1997.  
A semantics of ASMs with continuous time using infinitesimals is defined, their runs are described in some first order logic. The framework is used to discuss the verification of the railroad crossing problem, based upon its ASM specification in [253]. An early version appeared in 1996 as Technical Report 96-10 of Dept. of Informatics, Université Paris 12. For a continuation see [35].
35. D. Beauquier and A. Slissenko. A first-order logic for specification of timed algorithms: Basic properties and a decidable class. *Annals of Pure and Applied Logic*, 113(1–3):13–52, 2001.  
A continuation of [34]. The authors define (a) a class of algorithms (a modified version of ASMs) with explicit continuous time, and (b) a First-Order Timed Logic which suffices to write requirements specifications close to natural language, enhancing the logic in [34]. The timed logic description of the semantics of that class of ASMs can be viewed as a basic set of inductive invariants for proving the properties of timed ASMs. The authors consider a decidable class of verification problems and outline a compact verification proof of the Generalized Railroad Crossing Problem [253]. A first version of this work appeared under the title *On Semantics of Algorithms with Continuous Time* in October 1997 as TR 97-15 of Dept. of Informatics, Université Paris 12. A survey of that TR and of [34] appears under the title *Verification of Timed Algorithms: Gurevich Abstract State Machines versus First-Order Timed Logic* in Y. Gurevich and P. Kutter and M. Odersky and L. Thiele (eds.): *Abstract State Machines – ASM 2000*, *Int. Workshop on Abstract State Machines*, Monte Verita, Switzerland, Local Proceedings, March

2000, ETH Zürich, TIK-Report No. 87, pp. 22–39. Continuation in TR-00-23 of June 2000, Université Paris-12, by the same authors and with the title *A First Order Logic for Specification of Timed Algorithms: Basic Properties and a Decidable Class*. For a set of more efficient inductive invariants for ASMs and a short formal proof of the Generalized Railroad Crossing Problem along the same lines, see “A Predicate Logic Framework for Mechanical Verification of Real-Time Gurevich Abstract State Machines: A Case Study with PVS”, by the same authors together with T. Colard, Technical Report TR-00-25, Université Paris 12, Department of Informatics, 2000, available at <http://www.univ-paris12.fr/lacl/>.

36. B. Beckert and J. Pöschel. leanEA: A lean evolving algebra compiler. In H. K. Büning (ed.), *Proc. the Annual Conf. of the European Association for Computer Science Logic (CSL'95)*, Lecture Notes in Computer Science, Vol. 1092, pp. 64–85. Springer-Verlag, 1996.

A 9-line Prolog interpreter for sequential ASMs, including discussion of extensions for layered ASMs. A preliminary version appeared in April 1995 under the title *leanEA: A poor man's evolving algebra compiler* as internal report 25/95 of Fakultät für Informatik, Universität Karlsruhe.

37. P. Behm, P. Benoit, A. Faivre, and J. M. Meynadier. Meteor: A successful application of B in a large project. In *FM'99*, Lecture Notes in Computer Science, Vol. 1708, pp. 348–387. Springer-Verlag, 1999.

38. H. Behrends. *Beschreibung ereignisgesteuerter Aktivitäten in datenbankgestützten Informationssystemen*. PhD thesis, University of Oldenburg, Germany, 1995.

Uses ASMs to define the semantics of a language which is tailored to program the control of event-driven database applications. The specification proceeds by stepwise refinement of *event processing*, *rule selection* among the event triggered rules, and *action execution* following the priorities of the selected rules. Thesis supervised by Appelrath. Issued as TR 3/95 of CS Department of the University of Oldenburg, October 1995, 278 pages.

39. C. Beierle. Formal design of an abstract machine for constraint logic programming. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 377–382, Elsevier, Amsterdam, 1994.

Develops a general implementation scheme for CLP(X) over an unspecified constraint domain X, namely by designing a generic extension WAM(X) of the Warren Abstract Machine and a corresponding generic compilation scheme of CLP(X) programs to WAM(X) code. The scheme is based on the specification and correctness proof for compilation of Prolog programs in [132] and on joint work with Börger; see [40, 42, 41].

40. C. Beierle and E. Börger. Correctness proof for the WAM with types. In E. Börger, G. Jäger, H. Kleine Büning, and M. M. Richter (eds.), *Computer Science Logic*, Lecture Notes in Computer Science, Vol. 626, pp. 15–34. Springer-Verlag, 1992.

The specification and correctness proof for compiling Prolog to WAM [132] is extended in modular fashion to the type-constraint logic programming language Protos-L which extends Prolog with polymorphic order-sorted (dynamic) types. In this paper, the notion of types and dynamic type constraints are kept abstract (as constraints) in order to permit applications to different constraint formalisms like Prolog III or CLP(R). The theorem is proved that

for every appropriate type-constraint logic programming system L, every compiler to the WAM extension with an abstract notion of types which satisfies the specified conditions, is correct. Reference [41] extends the specification and the correctness proof to the full Protos Abstract Machine by refining the abstract type constraints to the polymorphic order-sorted types of PROTOS-L. Also issued as IBM Germany Science Center Research Report IWBS 205, 1991. Revised and final version published in [42].

41. C. Beierle and E. Börger. Refinement of a typed WAM extension by polymorphic order-sorted types. *Formal Aspects of Computing*, 8(5):539–564, 1996.  
Continuation of [42] which is extended to the full Protos Abstract Machine by refining the abstract type constraints to the polymorphic order-sorted types of PROTOS-L. Preliminary version published under the title *A WAM Extension for Type-Constraint Logic Programming: Specification and Correctness Proof* as Research Report IWBS 200, IBM Germany Science Center, Heidelberg, December 1991.

42. C. Beierle and E. Börger. Specification and correctness proof of a WAM extension with abstract type constraints. *Formal Aspects of Computing*, 8(4):428–462, 1996.

Revised version of [40].

43. C. Beierle, E. Börger, I. Durdanović, U. Glässer, and E. Riccobene. Refining abstract machine specifications of the steam boiler control to well documented executable code. In J.-R. Abrial, E. Börger, and H. Langmaack (eds.), *Formal Methods for Industrial Applications. Specifying and Programming the Steam-Boiler Control*, Lecture Notes in Computer Science, No. 1165, pp. 62–78. Springer-Verlag, 1996.

The steam-boiler control specification problem is used to illustrate how ASMs applied to the specification and the verification of complex systems can be exploited for a reliable and well-documented development of executable, but formally inspectable and systematically modifiable code. A hierarchy of stepwise refined abstract machine models is developed, the ground version of which can be checked for whether it faithfully reflects the informally given problem. The sequence of machine models yields various abstract views of the system, making the various design decisions transparent, and leads to a C++ program. This program has been demonstrated during the Dagstuhl Seminar on Methods for Semantics and Specification, in June 1995, to control the FZI Steam-Boiler simulator satisfactorily. The proofs of properties of the ASM models provide insight into the structure of the system which supports easily maintainable extensions and modifications of both the abstract specification and the implementation. For a continuation of this use of ASMs for reliable software development see [120, 125].

44. C. Beierle and G. Kern-Isberner. Modeling knowledge discovery and belief revision by Abstract State Machines. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 186–203. Springer-Verlag, 2003.

45. G. Bella and E. Riccobene. Formal analysis of the Kerberos authentication system. *J. Universal Computer Science*, 3(12):1337–1381, 1997.

A formal model of the whole system is reached through stepwise refinements of ASMs, and is used as a basis both to discover the minimum assumptions to guarantee the correctness of the system and to analyse its security weaknesses. Each refined model comes together with a correctness refinement theorem.

46. G. Bella and E. Riccobene. A realistic environment for crypto-protocol analyses by ASMs. In U. Glässer and P. Schmitt (eds.), *Proc. 5th Int. Workshop on Abstract State Machines*, pp. 127–138. Magdeburg University, 1998.  
ASMs are used to give a model of a general, realistic environment in which cryptographic protocols can be faithfully analyzed. The Needham–Schroeder protocol is investigated as an example.
47. A. Benczur, U. Glässer, and T. Lukovszki. Formal description of a distributed location service for mobile ad hoc networks. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 204–217. Springer-Verlag, 2003.  
Models a new routing layer protocol of mobile ad hoc networks by an async ASM, coming with sublayers for the location service and for the position based routing between known locations.
48. D. M. Berry. The importance of ignorance in requirements engineering. *J. Systems and Software*, 28(2):179–184, 1995.
49. G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.
50. J. Billington. Protocol specification using p-graphs, a technique based on coloured Petri nets. In W. Reisig and G. Rozenberg (eds.), *Lectures on Petri Nets II: Applications*, Lecture Notes in Computer Science, Vol. 1492. Springer-Verlag, 1998.
51. A. Binemann-Zdanowicz. Towards information systems modeling on the basis of ASM semantics. Technical Report 12/01, Brandenburg University of Technology at Cottbus, Germany, December 2001.  
Proposes an ASM based approach to modeling information services, using a special-purpose language *SiteLang*. Specifications which are written in *SiteLang* are compiled to input for XASM [15].
52. S. Bistarelli and E. Riccobene. An operational model for the SCLP language. ILPS Workshop on Tools and Environments for CLP held in Port Jefferson USA, 1997.  
Refinement and parallelization of the ASM model for Prolog to a semi-ring based constraint system, replacing the Call and Select rules of [71] by a Reduction rule which activates a child process simultaneously for each alternative of the current process. For the proceedings see [http://www.clip.dia.fi.upm.es/Tools\\_Environ/proceedings.html](http://www.clip.dia.fi.upm.es/Tools_Environ/proceedings.html).
53. B. Blakley. *A Smalltalk Evolving Algebra and its Uses*. PhD thesis, University of Michigan, Ann Arbor, Michigan, 1992.  
A reduced version of Smalltalk is formalized by sequential ASMs. A Hoare-style proof system is defined for reasoning about storage allocation and deallocation in ASMs. Missing constructs concern processes, inheritance, memory allocation and deallocation. Thesis supervised by Gurevich.
54. A. Blass. Abstract State Machines and pure mathematics. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 9–21. Springer-Verlag, 2000.  
A discussion of connections, similarities, and differences between concepts and issues arising in the study of ASMs and those of set theory and logic.

55. A. Blass, E. Börger, and Y. Gurevich. *Abstract State Machines*. Dagstuhl Seminar No. 02101, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, March 2002.
56. A. Blass and Y. Gurevich. The linear time hierarchy theorems for Abstract State Machines. *J. Universal Computer Science*, 3(4):247–278, 1997.
- Contrary to polynomial time, linear time depends on the computation model. In 1992, N. Jones designed specific computation models where the linear-speed-up theorem fails and linear-time computable functions form a proper hierarchy. In this paper linear-time hierarchy theorems for random access machines and ASMs are proven. In particular it is shown that there exists a lock-step universal sequential ASM  $U$ , i.e. with a constant  $c$  such that, under honest time counting,  $U$  simulates every other sequential ASM in lock-step with log factor  $c$ . The result has been announced under the title *Evolving Algebras and Linear Time Hierarchy* in B. Pehrson and I. Simon (eds.), IFIP 13th World Computer Congress, Vol. I: Technology/Foundations, Elsevier, Amsterdam, 1994, 383–390.
57. A. Blass and Y. Gurevich. Background, reserve, and Gandy machines. In P. Clote and H. Schwichtenberg (eds.), *Computer Science Logic (Proceedings of CSL 2000)*, Lecture Notes in Computer Science, Vol. 1862, pp. 1–17. Springer-Verlag, 2000.
- An investigation into the notion of the reserve set of an ASM, exploring the ideas of adding structure within the reserve (such as the hereditarily finite sets of [62]) and the non-determinism of importing new elements.
58. A. Blass and Y. Gurevich. The logic of choice. *J. Symbolic Logic*, 65(3):1264–1310, 2000.
- Motivated by the choice construct of ASMs, extensions of first-order logic with the choice construct (*choose*  $x : F(x)$ ) are studied. Some results about Hilbert’s  $\epsilon$  operator are proven. The main part of the paper concerns the case where all choices are independent. Previously appeared as Technical Report CSE-TR-369-98, EECS Dept., University of Michigan, 1998.
59. A. Blass and Y. Gurevich. New zero-one law and strong extension axioms. *Bull. EATCS*, 72:103–122, 2000.
- A formulation of Shelah’s proof of a zero-one law for the choiceless polynomial time variant of ASMs [62].
60. A. Blass and Y. Gurevich. Algorithms vs. machines. *Bull. EATCS*, 74:96–118, 2002.
- In reaction to [340] the mergesort algorithm is described in terms of distributed ASMs. See the description of recursive algorithms by turbo ASMs in [95].
61. A. Blass and Y. Gurevich. Abstract State Machines capture parallel algorithms. *ACM Trans. Computational Logic*, 2002, to appear.
- The axiomatization of sequential algorithms as the basis for a derivation of the sequential ASM thesis from the proposed axioms [249] is extended to parallel synchronous algorithms. A preliminary version appeared in November 2001 as Microsoft Research Technical Report TR-2001-117. See the adaptation to quantum algorithms in [234].
62. A. Blass, Y. Gurevich, and S. Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100:141–187, 1999.
- The question “Is there a computation model whose machines do not distinguish between isomorphic structures and compute exactly polynomial time

properties?” became a central question of finite model theory. The negative answer was conjectured in [244]. A related question is what portion of PTIME can be naturally captured by a computation model (when inputs are arbitrary finite structures). A PTIME version of ASMs is used to capture the portion of PTIME where algorithms are not allowed arbitrary choice but parallelism is allowed and, in some cases, implements choice. Earlier versions appeared as Technical Report CSE-TR-338-97, EECS Department, University of Michigan, 1997, and Technical Report MSR-TR-99-08, Microsoft Research, February 1999. See [235].

63. A. Blass, Y. Gurevich, and S. Shelah. On polynomial time computation over unordered structures. *J. Symbolic Logic*, 67(3):1093–1125, 2001.

A consideration of several algorithmic problems near the border of the known, logically defined complexity classes contained in polynomial time, including the choiceless polynomial time defined in [62].

64. A. Blass, Y. Gurevich, and J. Van den Bussche. Abstract State Machines and computationally complete query languages. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 22–33. Springer-Verlag, 2000.

The use of the choiceless polynomial-time variant of ASMs [62] as a query language for relational databases is explored. Also appears in TIK-Report No. 87, pp. 40–65, ETH Zürich, March 2000, and as Microsoft Research Technical Report MSR-TR-99-95. Republished in *Information and Computation* 174(1):20–36, 2002.

65. R. Bloomfield, D. Craigen, F. Koob, M. Ullman, and S. Wittmann. Formal methods diffusion: Past lessons and future prospects. In *Proc. SAFECOMP 2000*, Lecture Notes in Computer Science, Vol. 1943, pp. 211–226. Springer-Verlag, 2000.

The full technical report is available at [http://www.bsi.bund.de/aufgaben/projekte/fmethode/sonstige/fms\\_v1.0.pdf](http://www.bsi.bund.de/aufgaben/projekte/fmethode/sonstige/fms_v1.0.pdf).

66. C. Böhm and G. Jacopini. Flow diagrams, Turing Machines, and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, 1966.

67. T. Bolognesi and E. Börger. Abstract State Processes. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 22–32. Springer-Verlag, 2003.

Process-algebraic structuring techniques and concurrency patterns are combined with the state-based abstraction mechanism and synchronous parallelism of ASMs. Extended ASM programs are defined which are structured and evolve like process-algebraic behaviour expressions operating on evolving states. This supersedes the preceding definition given in the extended abstract presented to the Dagstuhl Seminar on ASMs in March 2002.

68. T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Comput. Netw. and ISDN Syst.*, 14(1):25–59, 1987.

69. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, 1999.

70. E. Börger. *Computability, Complexity, Logic (English translation of “Berechenbarkeit, Komplexität, Logik”)*, Studies in Logic and the Foundations of Mathematics, Vol. 128. North-Holland, 1989.

71. E. Börger. A logical operational semantics for full Prolog. Part I: Selection core and control. In E. Börger, H. Kleine Büning, M. M. Richter, and W. Schönfeld (eds.), *CSL'89. 3rd Workshop on Computer Science Logic*, Lecture Notes in Computer Science, Vol. 440, pp. 36–64. Springer-Verlag, 1990.  
See Comments to [74].
72. E. Börger. A logical operational semantics of full Prolog. Part II: Built-in predicates for database manipulation. In B. Rovin (ed.), *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 452, pp. 1–14. Springer-Verlag, 1990.  
See Comments to [74].
73. E. Börger. Dynamische Algebren und Semantik von Prolog. In E. Börger (ed.), *Berechenbarkeit, Komplexität, Logik*, pp. 476–499. Vieweg, 3rd edn., 1992.  
The first textbook definition of ASMs, elaborating notes of a series of lectures on the Semantics of Programming Languages delivered to a summer school organized by Börger in Cortona in 1989. The definition is illustrated with machines operating on standard data structures and by the tree based version [75] of the core Prolog ASM in [71].
74. E. Börger. A logical operational semantics for full Prolog. Part III: Built-in predicates for files, terms, arithmetic and input-output. In Y. N. Moschovakis (ed.), *Logic From Computer Science*, Berkeley Mathematical Sciences Research Institute Publications, Vol. 21, pp. 17–50. Springer-Verlag, 1992.  
This paper, along with [71] and [72] are the original 3 papers which provide a complete ASM formalization of Prolog with all features discussed in the international Prolog standardization working group (WG17 of ISO/IEC JTC1 SC22); see [101]. The specification proposed as the ground model for Prolog is developed by stepwise refinement, describing orthogonal language features by modular rule sets. An improved (tree instead of stack based) version is found in [75, 73, 131]. These three papers were also published in 1990 as IBM Germany Science Center Research Reports 111, 115 and 117 respectively. The refinement technique, used in combination with corresponding methods of proof, is further developed in [132, 133, 114, 42, 41, 104, 119, 136, 120, 138, 406, 387]. For a systematic exposition and survey see [94]. Together with the technique of building ground models, ASM refinements became a constituent of the ASM method.
75. E. Börger. A natural formalization of full Prolog. *Newsletter of the Association for Logic Programming*, 5(1):8–9, 1992.  
The paper explains the abstract tree structure and the four ASM transition rules which govern the user-defined core of Prolog. See [74].
76. E. Börger. Logic programming: The Evolving Algebra approach. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 391–395, Elsevier, Amsterdam, 1994.  
Surveys the work which has been done from 1988–1994 on specifications of logic programming systems by ASMs.
77. E. Börger. Review of E. W. Dijkstra and C. S. Scholten *Predicate Calculus and Program Semantics* (Springer-Verlag 1989). *Science of Computer Programming*, 23:1–11, 1994.  
Discusses the weakness of identifying the notion of proof with “formal proofs” and furthermore with “formal proofs in a strict format”. Critically evaluates the authors’ restricted view on the role of formal methods for program design

- and verification concerns. An abridged version appeared in *J. Symbolic Logic* 59:673–678, 1994.
78. E. Börger. Annotated bibliography on Evolving Algebras. In E. Börger (ed.), *Specification and Validation Methods*, pp. 37–51. Oxford University Press, 1995.
- An annotated bibliography of papers (as of 1994) which deal with or use ASMs. For an updated version see [116].
79. E. Börger. *Specification and Validation Methods*. Oxford University Press, 1995.
- The ASM related papers appearing in this volume are [248, 78, 133, 420, 114, 283, 256].
80. E. Börger. Why use Evolving Algebras for hardware and software engineering? In M. Bartosek, J. Staudek, and J. Wiederman (eds.), *Proc. SOFSEM'95, 22nd Seminar on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science, Vol. 1012, pp. 236–271. Springer-Verlag, 1995.
- A presentation of the salient features of ASMs, as part of a discussion and survey of the use of ASMs in the design and analysis of hardware and software systems. The leading example is detailed and improved in [119].
81. E. Börger. Evolving Algebras and Parnas tables. In H. Ehrig, F. von Henke, J. Meseguer, and M. Wirsing (eds.), *Specification and Semantics*. Dagstuhl Seminar No. 9626, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, July 1996.
- Extended abstract showing that Parnas' approach to the use of function tables for precise program documentation can be generalized in a natural way by using ASMs for well-documented program development.
82. E. Börger. Remarks on the history and some perspectives of Abstract State Machines in software engineering. In W. Aspray, R. Keil-Slawik, and D. L. Parnas (eds.), *The History of Software Engineering*, pp. 12–17. Dagstuhl Seminar No. 9635, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, August 1996.
- Survey of the development of the ASM method as of 1996. For an update in 2000 see [87].
83. E. Börger. How to use Abstract State Machines in software engineering. In S. Jähnichen, J. Loeckx, D. Smith, and M. Wirsing (eds.), *Logic for Systems Engineering*, Vol. 171, pp. 5–7. Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, 3–7 March 1997.
- The talk which triggered the first two years of work on the Java/JVM ASM project, as a comparative field test of purely declarative (functional or axiomatic) methods and their enhancement within an integrated abstract state-based operational (ASM) framework [138, 137, 139, 140, 141]. See preface to [406].
84. E. Börger. JUCS Special ASM Issue. Part II. In E. Börger (ed.), *J. Universal Computer Science*, Vol. 3(5). Springer-Verlag, 1997.
- Introduction to the second part of the special ASM issue of the *J. Universal Computer Science*. This May issue contains [311, 310, 439, 21, 120, 332, 424].
85. E. Börger. Ten Years of Gurevich's Abstract State Machines. In E. Börger (ed.), *J. Universal Computer Science*, Vol. 3(4). Springer-Verlag, 1997.
- Introduction to the first special ASM issue of the *J. Universal Computer Science*. This April issue contains [261, 56, 177, 409, 260, 313, 388].

86. E. Börger. High-level system design and analysis using Abstract State Machines. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann (eds.), *Current Trends in Applied Formal Methods (FM-Trends 98)*, Lecture Notes in Computer Science, Vol. 1641, pp. 1–43. Springer-Verlag, 1999.
- A general introduction to and survey of the ASM method, including the definition of the ASM concept and an illustration of the main characteristics of the method, a comparison with other well-known system design and analysis approaches, and experimental evidence for the ASM thesis.
87. E. Börger. Abstract State Machines at the cusp of the millenium. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 1–8. Springer-Verlag, 2000.
- A brief survey of the history of the development of the ASM method and the current challenges in the field (continuation of [82]).
88. E. Börger. *Hardware Design and Validation Methods*. Springer-Verlag, 2000.
- The ASM related paper appearing in this volume is [140].
89. E. Börger. Design for reuse via structuring techniques for ASMs. In R. Moreno-Díaz, B. Buchberger, and J.-L. Freire (eds.), *Computer Aided Systems Theory—EUROCAST 2001*, Lecture Notes in Computer Science, Vol. 2178, pp. 20–35. Springer-Verlag, 2001.
- The composition and structuring concepts for sequential ASMs defined in [134] are used to illustrate a modular high-level definition of the architecture of the Java Virtual Machine, unfolding its language layering and its functional components for loader, verifier, and interpreter. Extracted from [406].
90. E. Börger. Discrete systems modeling. In R. A. Meyers (ed.), *Encyclopedia of Physical Science and Technology*, Vol. 4, pp. 535–546. Academic Press, San Diego, 2001.
- A classification of discrete systems and of methods for their mathematical verification and experimental validation, using ASMs as the framework for the taxonomy.
91. E. Börger. Computation and specification models. A comparative study. In P. D. Mosses (ed.), *Proc. 4th Int. Workshop on Action Semantics*, BRICS Series, Vol. NS-02-8, pp. 107–130. Department of Computer Science at University of Aarhus, December 2002.
- Continuing the work in [81, 86, 142] representative computation models in the literature are characterized as naturally arising special classes of ASMs. Classical automata (Moore-Mealy, Co-Design FSM, Timed FSM, PushDown, Turing, Scott, Eilenberg, Minsky, Wegner, Alternating TM), grammar formalisms, tree computation machines, structured and functional programs are covered, as well as system design models like UNITY, COLD, B, SCR (Parnas tables), Petri nets, Neural Nets. Also logic based, functional-denotational and process-algebraic systems including CSP, Z, VDM are discussed. A draft has been presented under the title *Definitional Suggestions for Computation Theory* to the Dagstuhl Seminar “Theory and Application of Abstract State Machines”, March 3–8, 2002. The final version appears in [93].
92. E. Börger. The origins and the development of the ASM method for high-level system design and analysis. *J. Universal Computer Science*, 8(1):2–74, 2002.
- A historical and bibliographical survey of the ASM related literature from 1984–2002. Elaboration of [78, 82, 85, 116, 87].

93. E. Börger. Abstract State Machines: A unifying view of models of computation and of system design frameworks. *Annals of Pure and Applied Logic*, 2003, to appear.  
An elaboration of [91].
94. E. Börger. The ASM refinement method. *Formal Aspects of Computing*, 14, 2003, to appear.  
An exposition of the ASM refinement method, including a survey of its development in [75, 73, 131, 132, 133, 114, 42, 41, 104, 119, 136, 120, 138, 406, 387].
95. E. Börger and T. Bolognesi. Remarks on turbo ASMs for computing functional equations and recursion schemes. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003 – Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 218–228. Springer-Verlag, 2003.  
The notation for value-returning turbo ASMs defined in [134] is extended to simultaneous calls of multiple submachines and used to answer the question raised in [340] of how to naturally model widely used forms of recursion by abstract machines. The notation allows one to seamlessly integrate functional description and programming techniques into ASMs.
96. E. Börger, H. Busch, J. Cuellar, P. Päppinghaus, E. Tiden, and I. Wildgruber. Konzept einer hierarchischen Erweiterung von EURIS. Siemens ZFE T SE 1 Internal Report BBCPTW91-1 (pp. 1–43), Summer 1996.  
ASMs are proposed for extending the EURIS method for the tool-supported design of railway-related software.
97. E. Börger, G. D. Castillo, P. Glavan, and D. Rosenzweig. Towards a mathematical specification of the APE100 architecture: the APESE model. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 396–401, Elsevier, Amsterdam, 1994.  
Defines an ASM model of the high-level programmer’s view of the APE100 parallel architecture. This model is refined in [102] to an ASM processor model.
98. E. Börger, A. Cavarra, and E. Riccobene. An ASM semantics for UML activity diagrams. In T. Rus (ed.), *Algebraic Methodology and Software Technology, 8th Int. Conf., AMAST 2000, Iowa City, Iowa, USA, May 20-27, 2000 Proceedings*, Lecture Notes in Computer Science, Vol. 1816, pp. 293–308. Springer-Verlag, 2000.  
ASMs are used to disambiguate the semantics for activity diagrams in UML, defining a special subclass of ASMs appropriate to modeling such diagrams. As illustration a one-page UML activity diagram definition is given for the ASM model of Occam which appeared in [105]. For a continuation of this work to make semantic features of UML precise see [99].
99. E. Börger, A. Cavarra, and E. Riccobene. Modeling the dynamics of UML state machines. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 223–241. Springer-Verlag, 2000.  
The work in [98] providing a rigorous semantics for basic UML features is extended by an ASM definition of the dynamic semantics of UML state machines. These machines integrate statecharts with the UML object model. A rational reconstruction is given for the event-driven run to completion scheme of UML (including the sequential entry/exit actions, the concurrent internal

activities, and the event-deferring mechanism) and for the concepts of action and durative action. The models make the *semantic variation points* of UML explicit, as well as various ambiguities and omissions in the official UML documents. For an executable version of these models see [152], where various conflict situations are also described which may arise through the concurrent behavior of active objects. This argument has been reconsidered by the same authors in *Solving Conflicts in UML State Machines Concurrent States* presented to the Workshop on Concurrency Issues in UML – UML 2001, Toronto/Canada, and in *A precise semantics of UML State Machines: Making Semantic Variation Points and Ambiguities Explicit* in Proc. Int. Workshop on Semantic Foundations of Engineering Design Languages (SFEDL'02) in conjunction with the 5th European Joint Conferences on Theory and Practice of Software (ETAPS'02). See the continuation in [153].

100. E. Börger, M. Cesaroni, M. Falqui, and T. L. Murgi. Caso di Studio: Mail From Form System. Internal Report FST-2-1-RE-02, Fabbrica Servizi Telematici FST (Gruppo Atlantis), Uta (Cagliari), 1999.

Feasibility study of using ASMs for software analysis and design in an industrial object-oriented software development environment. Two company internal case studies are developed. In view of a possible integration, the use of the ASM method for building ground models and refining them to code is compared to the use of UML based tools, in particular Rational Rose.

101. E. Börger and K. Dässler. Prolog: DIN papers for discussion. ISO/IEC JTC1 SC22 WG17 Prolog Standardization Document 58, National Physical Laboratory, Middlesex, England, 1990.

A version of [71, 72, 74] proposed to the International Prolog Standardization Committee as a complete formal semantics of Prolog. A streamlined version is in [131], representing the definition of the dynamic core of Prolog which has been accepted as the ISO standard [291].

102. E. Börger and G. Del Castillo. A formal method for provably correct composition of a real-life processor out of basic components (The APE100 Reverse Engineering Study). In B. Werner (ed.), *Proc. 1st IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS'95)*, pp. 145–148, November 1995.

Presents an ASM based technique by which a behavioural description of a processor is obtained as the result of the composition of its (formally specified) basic architectural components. The technique is illustrated by the example of a subset of the zCPU processor (used as the control unit of the APE100 parallel architecture). A more complete version, containing the full formal description of the zCPU components, of their composition and of the whole zCPU processor, appeared in Y. Gurevich and E. Börger (eds.), *Evolving Algebras – Mini-Course, BRICS Technical Report (BRICS-NS-95-4)*, 195–222, University of Aarhus, Denmark, July 1995. This work is based upon G. Del Castillo's Tesi di Laurea “Descrizione Matematica dell'Architettura Parallela APE100”, Università di Pisa, academic year 1993/94. An extended abstract “An Evolving Algebra model for the APE100 parallel architecture” was presented to the 3d Annual Meeting of the Working Group 0.1.6 “Logik in der Informatik” of the German Association for Computer Science, University of Karlsruhe CS TR 23/95, pp. 48–51.

103. E. Börger and B. Demoen. A framework to specify database update views for Prolog. In M. J. Maluszynski (ed.), *PLILP'91. Third Int. Sympos. on Programming Languages Implementation and Logic Programming.*, Lecture Notes in Computer Science, Vol. 528, pp. 147–158. Springer-Verlag, 1991.

Provides a precise definition of the major Prolog database update views (immediate, logical, minimal, maximal), within a framework closely related to [71, 72, 74]. A preliminary version of this was published as *The View on Database Updates in Standard Prolog: A Proposal and a Rationale* in ISO/ETC JTC1 SC22 WG17 Prolog Standardization Report no. 74, February 1991, pp. 3–10.

104. E. Börger and I. Durdanović. Correctness of compiling Occam to Transputer code. *Computer Journal*, 39(1):52–92, 1996.

The final draft version has been issued in BRICS Technical Report (BRICS-NS-95-4); see [250]. It sharpens the refinement method of [132] to cope also with parallelism and non-determinism for an imperative programming language. The paper provides a mathematical definition of the Transputer instruction-set architecture for executing Occam together with a correctness proof for a general compilation schema of Occam programs into Transputer code.

Starting from the Occam model developed in [105], constituted by an abstract processor running a high- and a low-priority queue of Occam processes (which formalizes the semantics of Occam at the abstraction level of atomic Occam instructions), increasingly more refined levels of Transputer semantics are developed, proving correctness (and when possible also completeness) for each refinement step.

Along the way proof assumptions are collected, thus obtaining a set of natural conditions for compiler correctness, so that the proof is applicable to a large class of compilers. The formalization of the Transputer instruction-set architecture has been the starting point for applications of the ASM refinement method to the modeling of other architectures (see [102, 119]).

105. E. Börger, I. Durdanović, and D. Rosenzweig. Occam: Specification and compiler correctness. Part I: Simple mathematical interpreters. In U. Montanari and E. R. Olderog (eds.), *Proc. PROCOMET'94 (IFIP Working Conf. on Programming Concepts, Methods and Calculi)*, pp. 489–508. North-Holland, 1994.

Improving upon the parse tree determined ASM in [257], a truly concurrent ASM model of Occam is defined as the basis for a proven-to-be-correct, smooth transition to the Transputer instruction-set architecture. This model is stepwise refined, in a proven-to-be-correct way, providing: (a) an asynchronous implementation of synchronous channel communication, (b) its optimization for internal channels, (c) the sequential implementation of processors using priority and time-slicing. See [104] for the extension of this work to cover the compilation to Transputer code.

106. E. Börger, A. Gargantini, and E. Riccobene (eds.). *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589. Springer-Verlag, 2003.

This is the Proc. 10th Int. ASM Workshop (Taormina March 2003).

107. E. Börger and U. Glässer. A formal specification of the PVM architecture. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 402–409, Elsevier, Amsterdam, 1994.

After Börger's lectures on ASMs at the University of Paderborn in the early summer of 1993, Glässer suggested providing an ASM model for the Parallel Virtual machine (PVM [214], the Oak Ridge National Laboratory software system that serves as a general-purpose environment for heterogeneous distributed computing). The model in this paper defines PVM at the C-interface,

at the level of abstraction which is tailored to the programmer's understanding. Cf. the survey *An abstract model of the parallel virtual machine (PVM)* presented at *7th Int. Conf. on Parallel and Distributed Computing Systems (PDCS'94)*, Las Vegas/Nevada, 5.-9.10.1994. See [108] for an elaboration of this paper.

108. E. Börger and U. Glässer. Modeling and Analysis of Distributed and Reactive Systems using Evolving Algebras. In Y. Gurevich and E. Börger (eds.), *Evolving Algebras – Mini-Course, BRICS Technical Report BRICS-NS-95-4*, pp. 128–153. University of Aarhus, Denmark, July 1995.

This is a tutorial introduction to the ASM approach to design and verification of complex computing systems. The salient features of the method are explained by showing how one can develop from scratch an easily understandable and transparent ASM model for PVM [214], the widespread virtual architecture for heterogeneous distributed computing.

109. E. Börger and U. Glässer. Abstract State Machines 2001: New developments and applications. In E. Börger and U. Glässer (eds.), *J. Universal Computer Science*, Vol. 7(11), pp. 914–917. Springer-Verlag, 2001.

Introduction to the third special ASM issue of JUCS, with papers selected from those submitted after the Int. ASM'2001 Workshop held in Las Palmas. This issue contains [262, 387, 405, 142, 194, 208, 391].

110. E. Börger and U. Glässer. Abstract State Machines Workshop 2001. In R. Moreno-Díaz and A. Quesada-Arencibia (eds.), *Formal Methods and Tools for Computer Science*, pp. 212–304. IUCTC Universidad de Las Palmas de Gran Canaria, 2001.

Abstracts of talks presented to the Int. ASM'2001 Workshop held in Las Palmas de Gran Canaria from February 13–19, 2001, as part of Eurocast 2001. See [109].

111. E. Börger, U. Glässer, and W. Müller. The semantics of behavioral VHDL'93 descriptions. In *EURO-DAC'94. European Design Automation Conference with EURO-VHDL'94*, pp. 500–505, Los Alamitos, California, 1994. IEEE Computer Society Press.

Provides a transparent but precise ASM definition of the signal behavior and time model of full *elaborated* VHDL'93. This includes guarded signals, delta and time delays, the two main propagation delay modes *transport* and *inertial*, and the three process suspensions (wait on/until/for). Shared variables, postponed processes and rejection pulses are covered. The work is extended in [112].

112. E. Börger, U. Glässer, and W. Müller. Formal definition of an abstract VHDL'93 simulator by ea-machines. In C. Delgado Kloos and P. T. Breuer (eds.), *Formal Semantics for VHDL*, pp. 107–139. Kluwer Academic Publishers, 1995.

Extends the work in [111] by including the treatment of variable assignments and of value propagation by ports. References [383, 380] extend the VHDL model to analog VHDL and to Verilog.

113. E. Börger and R. Gotzhein. The light control case study. *J. Universal Computer Science*, 6(7):580–585, 2000.

The introductory pp. 580–585 present the requirements engineering case study, discussed during a Dagstuhl Seminar on Requirements Engineering [115], and a synopsis of the six solutions published in the journal issue. For the solution which uses ASMs see the comment to [125].

114. E. Börger, Y. Gurevich, and D. Rosenzweig. The bakery algorithm: Yet another specification and verification. In E. Börger (ed.), *Specification and Validation Methods*, pp. 231–243. Oxford University Press, 1995.

One ASM A1 is constructed to reflect faithfully the algorithm. Then a more abstract ASM A2 is constructed. It is checked that A2 is safe and fair, and that A1 correctly implements A2. The proofs work for atomic as well as for, *mutatis mutandis*, durative actions. See also [157, 258].

115. E. Börger, B. Hörger, D. L. Parnas, and D. Rombach. *Requirements Capture, Documentation, and Validation*, Vol. 241. Dagstuhl Seminar No. 99241, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, June 1999.

The Light Control Case Study was proposed to the participants of the seminar to discuss methods for solving requirements engineering problems. See [113] for a detailed exposition of some of the proposed solutions, including [125].

116. E. Börger and J. Huggins. Abstract State Machines 1988–1998: Commented ASM bibliography. *Bull. EATCS*, 64:105–127, 1998.

The 1997 version of the annotated bibliography of papers which deal with or use ASMs. An update of [78].

117. E. Börger, P. Joannou, and D. L. Parnas. *Practical Methods for Code Documentation and Inspection*, Vol. 178. Dagstuhl Seminar No. 9720, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, May 1997.

118. E. Börger, F. J. López-Fraguas, and M. Rodríguez-Artalejo. A model for mathematical analysis of functional logic programs and their implementations. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 410–415, Elsevier, Amsterdam, 1994.

Defines an ASM model for the innermost version of the functional logic programming language BABEL, extending the Prolog model of [131] by rules which describe the reduction of expressions to normal form. The model is stepwise refined towards a mathematical specification of the implementation of Babel by a graph-narrowing machine. The refinements are proved to be correct. A full version containing optimizations and proofs appeared under the title *Towards a Mathematical Specification of a Narrowing Machine* as research report DIA 94/5, Dpto. Informática y Automática, Universidad Complutense, Madrid 1994.

119. E. Börger and S. Mazzanti. A practical method for rigorously controllable hardware design. In J. P. Bowen, M. B. Hinchey, and D. Till (eds.), *ZUM'97: The Z Formal Specification Notation*, Lecture Notes in Computer Science, Vol. 1212, pp. 151–187. Springer-Verlag, 1997.

A technique for specifying and verifying the control of pipelined microprocessors is described, illustrated through ASM models for Hennessy and Paterson's RISC architecture DLX. A serial DLX model is stepwise refined to the parallel DLX with five-stage pipeline which is proved to be correct. Each refinement deals with a different pipelining problem (structural hazards, data hazards, control hazards) and the methods for its solution. This makes the approach applicable to design-driven verification as well as to the verification-driven design of RISC machines. A preliminary version appeared under the title *A correctness proof for pipelining in RISC architectures* as DIMACS (Rutgers University, Princeton University, AT&T Bell Laboratories, Bellcore) research report TR 96-22, pp. 1–60, Brunswick, New Jersey, July 1996. The

specification was worked out in 1994/95 by S. Mazzanti for her Tesi di Laurea *Algebra Dinamiche per il DLX*, Università di Pisa, 1995, supervised by Börger. For a machine verification using KIV and PVS of the refinement of the serial to the parallel model see [217, 407]. An omission in the proof for the second refinement step has been pointed out by Holger Hinrichsen, see [279] and Sect. 3.2 of this book for a correction. The specification and proof method has been applied in [286] to the commercial ARM2 RISC Microprocessor with a simpler three-stage pipeline and enhanced in [412] to automatically transform register transfer descriptions of microprocessors into executable ASMs.

120. E. Börger and L. Mearelli. Integrating ASMs into the software development life cycle. *J. Universal Computer Science*, 3(5):603–665, 1997.

Presents a structured software engineering method which allows the software engineer to control efficiently the *modular development* and the *maintenance* of well documented, formally inspectable and easily modifiable code out of rigorous ASM *models for requirement specifications*. Shows that the code properties of interest (like correctness, safety, liveness and performance conditions) can be proved at high levels of abstraction by traditional and reusable mathematical arguments which – where needed – can be computer verified. Shows also that the proposed method is appropriate for dealing in a rigorous but transparent manner with hardware–software co-design aspects of system development.

The approach is illustrated by developing a C++ program for the production-cell case study. The program has been validated by extensive experimentation with the FZI production cell simulator in Karlsruhe and has been submitted for inspection to the Dagstuhl Seminar on “Practical Methods for Code Documentation and Inspection” [117]. Source code (the ultimate refinement) for the case study appears in [332]; model checking results for the ASM models appear in [424] and in [362], where an error was detected in a refinement step for the deposit belt, due to an erroneous assumption of symmetry between unloading actions for feed belt, press and deposit belt. For a PVS verification of the case see [207]. An abstract appeared under the title “The Evolving Algebra Approach to Modular Development of Well Documented Software for Complex Systems. A Case Study: The Production Cell Control Program” in the Proc. DIMACS Workshop on Controllers for Manufacturing and Automation: Specification, Synthesis, and Verification Issues–CONMASSYV, May 1996, DIMACS. The work was part of Mearelli’s Tesi di Laurea *Sviluppo Sistematico di un Programma di Controllo per un Impianto di Produzione Robotizzato*, Pisa 1994/95, supervised by Börger.

121. E. Börger, P. Pöppinghaus, and J. Schmid. Report on a practical application of ASMs in software design. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 361–366. Springer-Verlag, 2000.

A report on the successful use of ASMs at Siemens AG (from May 1998 to March 1999) to redesign and implement the railway process model component of FALKO, a railway timetable validation and construction program. Extensive testing was done for the ASM model prior to its coding, using the ASM Workbench [170]. For a powerpoint slide show see *Falko* ( $\leadsto$  CD).

122. E. Börger and E. Riccobene. A mathematical model of concurrent Prolog. Research Report CSTR-92-15, Dept. of Computer Science, University of Bristol, Bristol, England, 1992.

An ASM formalization of Ehud Shapiro’s Concurrent Prolog. Adaptation of the model defined for PARLOG in [123].

123. E. Börger and E. Riccobene. A formal specification of Parlog. In M. Droste and Y. Gurevich (eds.), *Semantics of Programming Languages and Model Theory*, pp. 1–42. Gordon and Breach, 1993.
- An ASM formalization of Parlog, a well-known parallel version of Prolog. This formalization separates explicitly the two kinds of parallelism occurring in Parlog: AND-parallelism and OR-parallelism. It uses an implementation-independent, abstract notion of terms and substitutions and is obtained by combining the concurrent features of the Occam model of [257] with the logic programming model of [75]. Also published as Technical Report TR 1/93 from Dipartimento di Informatica, Università di Pisa, 1993. An improved and extended version of the following two papers by the same authors: *Logical Operational Semantics of Parlog. Part I: And-Parallelism* in H. Boley and M. M. Richter (eds.): *Processing Declarative Knowledge (Lecture Notes in Artificial Intelligence, Vol. 567)*, pp. 191–198, Springer-Verlag, 1991). *Logical Operational Semantics of Parlog. Part II: Or-Parallelism* in A. Voronkov (ed.): *Logic Programming (Lecture Notes in Artificial Intelligence, Vol. 592)*, pp. 27–34, Springer-Verlag, 1992). For an extension to Pandora see [374].
124. E. Börger and E. Riccobene. Logic + control revisited: An abstract interpreter for Gödel programs. In G. Levi (ed.), *Advances in Logic Programming Theory*, pp. 231–154. Oxford University Press, 1994.
- Develops a simple ASM interpreter for Gödel programs. This interpreter abstracts from the deterministic and sequential execution strategies of Prolog [132] and thus provides a precise interface between logic and control components for execution of Gödel programs. The construction is given in abstract terms which cover the general logic programming paradigm and allow for concurrency.
125. E. Börger, E. Riccobene, and J. Schmid. Capturing requirements by Abstract State Machines: The light control case study. *J. Universal Computer Science*, 6(7):597–620, 2000.
- ASMs are applied to the Light Control Case Study discussed during a Dagstuhl Seminar on Requirements Engineering [115]. A ground model is defined which captures the informal requirements as far as possible and documents their ambiguity and incompleteness. The ground model is then refined into a form directly executable by AsmGofer [390].
126. E. Börger and D. Rosenzweig. An analysis of prolog database views and their uniform implementation. ISO/IEC JTC1 SC22 WG17 Prolog Standardization Document 80, National Physical Laboratory, Teddington, Middlesex, England, 1991.
- A mathematical analysis of the Prolog database views defined in [103]. The analysis is derived by stepwise refinement of the stack model for Prolog from [132]. It leads to the proposal of a uniform implementation of the different views which discloses the tradeoffs between semantic clarity and efficiency of database update view implementations. Also issued as Research Report CSE-TR-89-91 by the EECS Dept., University of Michigan, Ann Arbor.
127. E. Börger and D. Rosenzweig. A formal specification of Prolog by tree algebras. In V. Čeric, V. Dobrić, V. Lužar, and R. Paul (eds.), *Information Technology Interfaces*, pp. 513–518. University Computing Center, Zagreb, Zagreb, 1991.
- Prompted by discussion in the international Prolog standardization committee (ISO/IEC JTC1 SC22 WG17), this paper suggests replacing the stack-based model of [71] and the stack implementation of the tree-based model

of [72] by a pure tree model for Prolog. See also [75, 73], which is the basis for [131], where a mistake in the treatment of the *catch* built-in predicate is corrected.

128. E. Börger and D. Rosenzweig. From Prolog algebras towards WAM – a mathematical study of implementation. In E. Börger, H. Kleine Büning, M. M. Richter, and W. Schönfeld (eds.), *CSL'90, 4th Workshop on Computer Science Logic*, Lecture Notes in Computer Science, Vol. 533, pp. 31–66. Springer-Verlag, 1991.

Refines Börger's Prolog model [72] by elaborating the conjunctive component – as reflected by compilation of clause structure into WAM code – and the disjunctive component – as reflected by compilation of predicate structure into WAM code. The correctness proofs for these refinements include last call optimization, determinacy detection and virtual copying of dynamic code. Extended in [129] and improved in [132].

129. E. Börger and D. Rosenzweig. WAM algebras – a mathematical study of implementation, Part 2. In A. Voronkov (ed.), *Logic Programming*, Lecture Notes in Artificial Intelligence, Vol. 592, pp. 35–54. Springer-Verlag, 1992.

Refines the Prolog model of [128] by elaborating the WAM code for representation and unification of terms. The correctness proof for this refinement includes environment trimming, Warren's variable classification and switching instructions. Improved in [132]. Also issued as Technical Report CSE-TR-88-91 from EECS Dept, University of Michigan, Ann Arbor, Michigan, 1991.

130. E. Börger and D. Rosenzweig. The mathematics of set predicates in Prolog. In G. Gottlob, A. Leitsch, and D. Mundici (eds.), *Computational Logic and Proof Theory*, Lecture Notes in Computer Science, Vol. 713, pp. 1–13. Springer-Verlag, 1993.

Provides a logical (proof-theoretical) specification of the solution-collecting predicates *findall*, *bagof* of Prolog. This abstract ASM-based definition allows a logico-mathematical analysis, rationale and criticism of various proposals made for implementations of these predicates (in particular of *setof*) in current Prolog systems. Foundational companion to Sect. 5, on solution collecting predicates, in [131]. Also issued as *Prolog. Copenhagen papers 2*, ISO/IEC JTC1 SC22 WG17 Standardization report no. 105, National Physical Laboratory, Middlesex, 1993, pp. 33–42.

131. E. Börger and D. Rosenzweig. A mathematical definition of full Prolog. *Science of Computer Programming*, 24:249–286, 1995.

An abstract ASM specification of the semantics of Prolog, rigorously defining the international ISO 1995 Prolog standard by stepwise refinement. Revised and final version of [71, 72, 101, 127, 75, 73]. An abstract of this was issued as *Full Prolog in a Nutshell* in *Logic Programming* (Proc. 10th Int. Conf. on Logic Programming) (D. S. Warren, Ed.), MIT Press 1993. A preliminary version appeared under the title *A Simple Mathematical Model for Full Prolog* as research report TR-33/92, Dipartimento di Informatica, Università di Pisa, 1992.

132. E. Börger and D. Rosenzweig. The WAM – definition and compiler correctness. In C. Beierle and L. Plümer (eds.), *Logic Programming: Formal Methods and Practical Applications*, Studies in Computer Science and Artificial Intelligence, Vol. 11, Chap. 2, pp. 20–90. North-Holland, 1995.

The successive-refinement method introduced for ASMs in [71, 72, 74] is applied to provide a hierarchy of models as a mathematical basis for constructing provably correct compilers from Prolog to WAM. Various refinement steps

take care of different distinctive features (“orthogonal components”) of WAM, making the specification as well as the correctness proof modular and extendible; examples of such extensions are found in [41, 42, 133, 21, 313]. An extension of this work to an imperative language with parallelism and non-determinism has been provided in [104] and is further developed in [137, 141, 406]. See [369, 388] for machine-checked versions of the correctness proofs for the refinement steps. Preliminary versions appeared in [128, 129] and as Research Report TR-14/92, Dipartimento di Informatica, Università di Pisa, 1992.

133. E. Börger and R. Salamone. CLAM specification for provably correct compilation of  $CLP(\mathcal{R})$  programs. In E. Börger (ed.), *Specification and Validation Methods*, pp. 97–130. Oxford University Press, 1995.

Extends the specification and correctness proof, for compiling Prolog programs to the WAM [132], to  $CLP(\mathcal{R})$  and the constraint logical arithmetical machine (CLAM) developed at IBM Yorktown Heights. For full proofs, see R. Salamone, “Una Specifica Astratta e Modulare della CLAM (An Abstract and Modular Specification of the CLAM)”, Tesi di Laurea, supervised by Börger at Università di Pisa, Italy, academic year 1992/93, pp. 113.

134. E. Börger and J. Schmid. Composition and submachine concepts for sequential ASMs. In P. Clote and H. Schwichtenberg (eds.), *Computer Science Logic (Proceedings of CSL 2000)*, Lecture Notes in Computer Science, Vol. 1862, pp. 41–60. Springer-Verlag, 2000.

Structuring concepts for sequential composition and iteration, parameterization, and encapsulation in ASMs are defined. The concept of recursive submachines has been developed for its use in [406] to provide a modular definition of the statics and the dynamics of Java and of the JVM architecture [89] which can be naturally refined to an executable model, namely written in AsmGofer [390].

135. E. Börger and P. Schmitt. A formal operational semantics for languages of type Prolog III. In E. Börger, H. Kleine Büning, M. M. Richter, and W. Schönfeld (eds.), *CSL '90, 4th Workshop on Computer Science Logic*, Lecture Notes in Computer Science, Vol. 533, pp. 67–79. Springer-Verlag, 1991.

An ASM formalization of Alain Colmerauer’s constraint logic programming language Prolog III, obtained from the Prolog model in [71, 72, 74] through extending unifications by constraint systems. This extension was the starting point for the extension of [132] in [40]. A preliminary version of this was issued as IBM Germany IWBS Report 144, 1990.

136. E. Börger and P. Schmitt. A description of the tableau method using Abstract State Machines. *J. Logic and Computation*, 7(5):661–683, 1997.

Starting from the textbook formulation of the tableau calculus, an operational description of the tableau method is given in terms of ASMs at various levels of refinement ending after four stages at a specification that is close to the lean  $T^AP$  implementation of the tableau calculus in Prolog. Proofs of correctness and completeness of the refinement steps are given.

137. E. Börger and W. Schulte. Defining the Java Virtual Machine as platform for provably correct Java compilation. In L. Brim, J. Gruska, and J. Zlatuska (eds.), *Mathematical Foundations of Computer Science 1998, 23rd Int. Sympos., MFCS'98, Brno, Czech Republic*, Lecture Notes in Computer Science, Vol. 1450, pp. 17–35. Springer-Verlag, August 1998.

A definition of the Java Virtual Machine, along with a provably correct compilation scheme for Java programs to the JVM, based on the ASM semantics

for Java presented in [138]. Streamlined, corrected and completed in [406]. The full version appears as Technical Report, Universität Ulm, Fakultät für Informatik, Ulm, Germany, 1998.

138. E. Börger and W. Schulte. Programmer friendly modular definition of the semantics of Java. In J. Alves-Foss (ed.), *Formal Syntax and Semantics of Java*, Lecture Notes in Computer Science, Vol. 1523. Springer-Verlag, 1998.

Provides a system- and machine-independent definition of the semantics of the full programming language Java as seen by the Java programmer. The definition is modular, coming as a series of refined ASMs, dealing in succession with Java's imperative core, its object-oriented features, exceptions and threads. Streamlined, corrected and completed in [406]. An extended abstract has been presented by Börger to the IFIP WG 2.2 (University of Graz, 22–26 September, 1997) and by Schulte under the title *Modular Dynamic Semantics of Java* to the Workshop on Programming Languages (Ahrensdoorp, FEHMARN Island, September 25, 1997), see University of Kiel, Dept. of CS Research Report Series, TR *Arbeitstagung Programmiersprachen* 1997. An independently developed Java model using ASMs and Montages was published later as a technical report in [421]. For an ASM model of Java which is geared to the analysis of the concurrency features see [259].

139. E. Börger and W. Schulte. Initialization problems for Java. *Software – Concepts and Tools*, 19(4):175–178, 2000.

Using the models in [138, 137] and reporting results of experiments with current implementations of the JVM it is shown that the treatment of initialization of classes and interfaces in Java and in the Java Virtual Machine do not match, afflicting the portability of Java programs. It is shown that concurrent initialization may deadlock and that various Java compilers violate the initialization semantics through standard optimization techniques.

140. E. Börger and W. Schulte. Modular design for the Java VM architecture. In E. Börger (ed.), *Architecture Design and Validation Methods*, pp. 297–357. Springer-Verlag, 2000.

Provides a modular definition of the Java VM architecture, at different layers of abstraction. The layers partly reflect the layers made explicit in the specification of the Java language in [138]. The ASM model for JVM defined here and the ASM model for Java defined in [138] provide a rigorous framework for a machine independent mathematical analysis of the language and of its implementation, including compilation correctness conditions, safety and optimization issues. Streamlined, corrected and completed in [406].

141. E. Börger and W. Schulte. A practical method for specification and analysis of exception handling: A Java/JVM case study. *IEEE Trans. Software Eng.*, 26(10):872–887, October 2000.

ASM models for exception handling in Java and the Java Virtual Machine (JVM) are given, along with a compilation scheme for Java to JVM code. It is proven that corresponding runs of the Java and JVM throw the same exceptions with equivalent effect. A different proof is offered in [406].

142. E. Börger and D. Sona. A neural abstract machine. *J. Universal Computer Science*, 7(11):1007–1024, 2001.

A parameterized Neural Abstract Machine is defined whose instantiations cover the major neural networks in the literature. The refinement for feedforward networks with back-propagation training is shown.

143. D. Bowen. Implementation at Quintus of Börger's Prolog ASM. Personal Communication to Börger at Quintus in Palo Alto on November 5 and e-mail of November 11, 1990.
- The four ASM rules which constitute the core for user-defined predicates in Börger's Prolog model [101, 75] have been implemented, making use of the code available at Quintus to compute the abstract functions which appear in that model, in particular the function *unify*, and the function *procdef* which for a given goal (literal) and a given program yields the ordered set of alternatives the program offers for resolving the goal.
144. M. Broy, S. Merz, and K. Spies. The RPC memory case study: A synopsis. In M. Broy, S. Merz, and K. Spies (eds.), *Formal Systems Specification – The RPC-Memory Specification Case Study*, Lecture Notes in Computer Science, Vol. 1169. Springer-Verlag, August 1996.
- For an ASM solution of the case study see [284].
145. A. Brüggemann, L. Priese, D. Rödding, and R. Schätz. Modular decomposition of automata. In E. Börger, G. Hasenjäger, and D. Rödding (eds.), *Logic and Machines: Decision Problems and Complexity*, Lecture Notes in Computer Science, Vol. 171, pp. 198–236. Springer-Verlag, 1984.
146. B. Buchberger and B. Roider. Input/output codings and transition functions in effective systems. *Int. J. General Systems*, 4:201–209, 1978.
147. J. R. Burch. Techniques for verifying superscalar microprocessors. In *Proc. 33rd Annual Conf. on Design Automation Conference*, pp. 552–557, Las Vegas, Nevada, 3–7 June 1996. ACM Press.
148. W. Burgard, A. B. Cremers, D. Fox, M. Heidelbach, A. M. Kappel, and S. Lüttringhaus-Kappel. Knowledge-enhanced CO-monitoring in coal mines. In *Proc. Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-AIE)*, pp. 511–521, Fukuoka, Japan, 4–7 June 1996.
- Extends the ASM interpreter of [301] by modules, which can be executed in parallel, so that distributed processes can be represented which are synchronized via stream communication. Also a graphical visualization is added, which is needed for industrial applications of the system in a time- and security-critical coal mining application reported in the paper. Available at <http://www.informatik.uni-bonn.de/~angelica/publications.html>.
149. C. Campbell and Y. Gurevich. Table ASMs. In R. Moreno-Díaz and A. Quesada-Arencibia (eds.), *Formal Methods and Tools for Computer Science (Local Proceedings of Eurocast 2001)*, pp. 286–290, Canary Islands, Spain, February 2001. Universidad de Las Palmas de Gran Canaria.
- A special table notation for a class of basic ASMs is presented.
150. G. D. Castillo and P. Päppinghaus. Designing software for internet telephony: experiences in an industrial development process. In A. Blass, E. Börger, and Y. Gurevich (eds.), *Theory and Applications of Abstract State Machines*, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, 2002.
- The development of a high-level abstract model of the core functionality of an entity in a mobile telephony network is reported. The model served as the basis for a C++ product implementation.
151. S. Cater and J. Huggins. An ASM dynamic semantics for standard ML. Technical Report CPSC-1999-2, Kettering University, Flint, Michigan, October 1999.

ASMs are used to provide dynamic semantics for the functional programming language Standard ML. An extended abstract appears in Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, eds., *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 203–222, Springer-Verlag, 2000, and in TIK-Report No. 87, pp. 68–99, ETH Zürich, March 2000.

152. A. Cavarra. *Applying Abstract State Machines to Formalize and Integrate the UML Lightweight Method*. PhD thesis, University of Catania, Sicily, Italy, 2000.

The thesis, which was supervised by Börger and Riccobene, studies the use of ASMs to rigorously support semi-formal specification techniques as they are used in industrial practice, with a focus on UML notations and concepts. In addition to the work, which has been published in [98, 99], a simulator for UML state machines has been developed using AsmGofer [390].

153. A. Cavarra, E. Riccobene, and P. Scandurra. Integrating UML static and dynamic views and formalizing the interaction mechanism of UML state machines. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 229–243. Springer-Verlag, 2003.
154. A. Cavarra, E. Riccobene, and A. Zavanella. A formal model for the parallel semantics of P3L. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim (eds.), *Proc. 2000 ACM Sympos. Applied Computing*, Lecture Notes in Computer Science, Vol. 2, pp. 804–812. ACM Press, 2000.

Provides an ASM formalization of the semantics of P3L, a programming language with task and data parallelism. The model describes (a) how the compiler defines a network of processes starting from a given program, and (b) the computation of the running processes. Some rewrite rules for trimming the compiler for better program performance are proved to be correct.

155. Z. Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison Wesley, 2000.

See also <http://java.sun.com/products/javacard/havacard21.html>.

156. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
157. J. Cohen and A. Slissenko. On verification of refinements of asynchronous timed distributed algorithms. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 34–49. Springer-Verlag, 2000.

A study of the role of timing constraints for proving the correctness of refinements of distributed asynchronous algorithms with continuous time, specified as distributed ASMs. The ASM investigation of Lamport's Bakery Algorithm in [114] is used as a case study. Also appears in TIK-Report No. 87, pp. 100–114, ETH Zürich, March 2000.

158. K. Compton, Y. Gurevich, J. Huggins, and W. Shen. An automatic verification tool for UML. Technical Report CSE-TR-423-00, EECS Department, University of Michigan, 2000.

Using the ideas developed in [98, 99, 152], ASMs are used to give semantics for UML state machines, as a basis for constructing an automated tool for verifying the properties of UML state machines. An extended abstract appears as “A Semantic Model for the State Machine in the Unified Modeling Language” in G. Reggio, A. Knapp, B. Rumpe, B. Selic, and R. Wieringa (eds.),

- “Dynamic Behaviour in UML Models: Semantic Questions”, Workshop Proceedings, UML 2000 Workshop, Ludwig-Maximilians-Universität München, Institut für Informatik, Bericht 0006, October 2000, pp. 25–31.
159. D. Craigen, S. Gerhart, and T. Ralston. Formal methods reality check: Industrial usage. *IEEE Trans. Software Eng.*, 21(2):90–98, 1995.
  160. A. B. Cremers, U. Griefahn, and R. Hinze. *Deduktive Datenbanken*. Vieweg, 1994.
  161. A. B. Cremers and T. N. Hibbard. Formal modeling of virtual machines. *IEEE Trans. Software Eng.*, SE-4(5):426–436, 1978.
  162. F. Cristian. Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Computing*, 6:175–187, 1991.
  163. F. DaCruz. *Kermit: A File Transfer Protocol*. Digital Press, 1987.  
See also the Kermit Web site <http://www.columbia.edu/kermit>.
  164. O. Dahl, E. W. Dijkstra, and C. A. R. Hoare. *Structured Programming*. Academic Press, 1972.
  165. W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
  166. M. Davis. *The Universal Computer: The Road from Leibniz to Turing*. W.W. Norton, New York, 2000.
  167. W. P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, Cambridge, 1998.
  168. M. Dehof and S. Tahar. Implementierung des DLX-RISC-Prozessors in einer Standardzellen-Entwurfsumgebung. Technical Report SBF 358-C2-9/94, Institute of Computer Design and Fault Tolerance, University of Karlsruhe, Germany, 1994.
  169. G. Del Castillo. Towards comprehensive tool support for Abstract State Machines. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann (eds.), *Applied Formal Methods – FM-Trends 98*, Lecture Notes in Computer Science, Vol. 1641, pp. 311–325. Springer-Verlag, 1999.  
A description of the ASM Workbench, an integrated environment for various ASM tools; see [170]. Another description appears under the title *The ASM Workbench: an Open and Extensible Tool Environment for Abstract State Machines* in [226, pp. 139–154].
  170. G. Del Castillo. *The ASM Workbench. A Tool Environment for Computer-Aided Analysis and Validation of Abstract State Machine Models*. PhD thesis, Universität Paderborn, Germany, 2001.  
Published in: HNI-Verlagsschriftenreihe Vol. 83, 217 pages. The main contribution of the thesis, supervised by Börger and Glässer, is the definition of the ASM-based specification language ASM-SL and a tool architecture – the ASM Workbench – based on ASM-SL. The tool environment includes basic functionalities such as parsing, abstract syntax trees, type checking, pretty printing, etc., and in particular a transformation of ASMs into FSMs which can be model checked using SMV; see [175]. In the thesis a case study from the domain of automated manufacturing is treated, namely the distributed control for a material flow system. The ASM Workbench has been extensively used for testing purposes in the FALKO project at Siemens [121]. It has been used in [355] to provide an executable semantics for UML.

171. G. Del Castillo, I. Durdanović, and U. Glässer. An evolving algebra abstract machine. In H. K. Büning (ed.), *Proc. Ann. Conf. of the European Association for Computer Science Logic (CSL'95)*, Lecture Notes in Computer Science, Vol. 1092, pp. 191–214. Springer-Verlag, 1996.

Introduces the concept of an abstract machine (EAM) as a platform for the systematic development of ASM tools and gives a formal definition of the EAM ground model in terms of a universal ASM. The definition proceeds by stepwise refinement and leads to the design of a simple virtual machine architecture as a basis for a sequential implementation of the EAM. A preliminary version appeared under the title *Specification and Design of the EAM (EAM – Evolving Algebra Abstract Machine)* as Technical Report TR-RSFB-96-003, Paderborn University, 1996.

172. G. Del Castillo and U. Glässer. Computer-aided analysis and validation of heterogeneous system specifications. In F. Pichler, R. Moreno-Díaz, and P. Kopacek (eds.), *Computer Aided Systems Theory: Proc. 7th Int. Workshop on Computer Aided Systems Theory (EUROCAST'99)*, Lecture Notes in Computer Science, Vol. 1798, pp. 55–79. Springer-Verlag, 2000.

ASMs are proposed as a method for combining heterogeneous specifications. As a case study, Petri-net and SDL specifications of a material flow system are combined via ASMs and validated using SMV [175].

173. G. Del Castillo and W. Hardt. Fast dynamic analysis of complex hardware/software systems based on Abstract State Machine models. In *Proc. 6th Int. Workshop on Hardware/Software Codesign (CODES/CASHE'98) (March 15–18, Seattle, Washington)*, pp. 77–81, 1998.

Provides experimental results for [174].

174. G. Del Castillo and W. Hardt. Towards a unified analysis methodology of HW/SW systems based on Abstract State Machines: Modeling of instruction sets. In *Proc. GI/ITG/GMM Workshop “Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen”*, Paderborn, Germany, March 1998.

Extending the processor description technique from [102], ASMs are used for high-level analysis of hardware/software systems. The authors show how to model instruction sets using ASMs and to instrument such models to collect data for evaluating design alternatives. Experimental results appear in [173].

175. G. Del Castillo and K. Winter. Model checking support for the ASM high-level language. In S. Graf and M. Schwartzbach (eds.), *Proc. 6th Int. Conf. TACAS 2000*, Lecture Notes in Computer Science, Vol. 1785, pp. 331–346. Springer-Verlag, 2000.

Extending [424], the authors introduce an interface from the ASM Workbench to the SMV model checking tool, based on an ASM-to-SMV transformation. Previously appeared as Universität-GH Paderborn Technical Report TR-RI-99-209. For an extension see [425, 426]. For an experiment with this interface see [172].

176. J. Derrick and E. Boiten. *Refinement in Z and Object-Z*. Formal Approaches to Computing and Information Technology. Springer-Verlag, 2001.

177. S. Dexter, P. Doyle, and Y. Gurevich. Gurevich Abstract State Machines and Schönhage Storage Modification Machines. *J. Universal Computer Science*, 3(4):279–303, 1997.

A demonstration that, in a strong sense, Schönhage’s storage modification machines are equivalent to unary basic ASMs without external functions.

The unary restriction can be removed if the storage modification machines are equipped with a pairing function in an appropriate way.

178. V. Di Iorio, R. Bigonha, and M. Maia. A self-applicable partial evaluator for ASM. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines – ASM 2000, Int. Workshop on Abstract State Machines, Monte Verità, Switzerland, Local Proceedings*, TIK-Report, No. 87, pp. 115–130. ETH Zürich, March 2000.

A partial evaluator for ASMs is described which is self-applicable. The use of such a tool for compiler generation and techniques for describing language semantics suitable for partial evaluation are discussed. Implementation details are in *An ASM Implementation of a Self-Applicable Partial Evaluator* by V. Di Iorio and R. Bigonha, Technical Report LLP-004-2000 of Programming Languages Laboratory, DCC, Universidade Federal de Minas Gerais, 2000. Extends the work of [252].

179. S. Diehl. Transformations of evolving algebras. In *Proc. LIRA '97 (VIII Int. Conf. on Logic and Computer Science)*, pp. 43–50, Novi Sad, Yugoslavia, September 1997.

Constant propagation is introduced as a transformation on ASMs. ASMs are extended by macro definitions, folding and unfolding transformations for macros, a simple transformation to flatten transition rules and a pass separation transformation for ASMs are defined. For all transformations the operational equivalence of the resulting ASMs with the original ASMs is proven. In the case of pass separation, it is shown that the results of the computations in the original and the transformed ASMs are equal. Pass separation is applied to a simple interpreter. A preliminary version appeared in 1995 as Technical Report 02/95 of Universität des Saarlandes.

180. D. Diesen. *Specifying Algorithms Using Evolving Algebra. Implementation of Functional Programming Languages*. D Sc Thesis, Dept. of Informatics, University of Oslo, Norway, March 1995.

A description of a functional interpreter for ASMs, with applications for functional programming languages, along with a proposed extension to the language of ASMs.

181. B. DiFranco. *Specification of ISO SQL using Montages*. Master's thesis, Università di l'Aquila, Italy, 1997.

Tesi di Laurea, in Italian.

182. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

183. E. W. Dijkstra. Structure of the T.H.E. multiprogramming system. *Commun. ACM*, 11:341–346, 1968.

184. E. W. Dijkstra. Notes on structured programming. In O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare (eds.), *Structured Programming*, pp. 1–82. Academic Press, 1972.

185. E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

186. S. Distefano. *Architettura X86 e sistema dei processi di MINIX dalla specifica ASM al codice eseguibile*. Master's thesis, University of Catania, Sicily, Italy, 1998/99.

The multiprogramming operating system MINIX together with the architecture X86 are modeled as ASMs and refined to an implementation in Java and in C.

187. A. Dold. A formal representation of Abstract State Machines using PVS. Verifix Technical Report Ulm/6.2, Universität Ulm, Germany, July 1998.
- A technique for formally representing ASMs using the automated verification system PVS is described, along with generic PVS theories which define refinement relations between ASMs. An application to *Representing the Alpha Processor Family using PVS* by the same author appears as Verifix/Uni Ulm/4.1, University of Ulm, Germany, November 1995.
188. A. Dold, T. Gaul, V. Vialard, and W. Zimmermann. ASM-based mechanized verification of compiler back-ends. In U. Glässer and P. Schmitt (eds.), *Proc. 5th Int. Workshop on Abstract State Machines*, pp. 50–67. Magdeburg University, 1998.
- Using techniques from [439], an approach is described for mechanically proving the correctness of back-end rewrite system (BURS) specifications where source and target languages are described by ASMs. The approach can be used in conjunction with BURS-based back-end compiler generators. PVS proof strategies are defined for the automatic verification of BURS rules. Similar aspects are treated by A. Dold and T. Gaul and W. Zimmermann in *Mechanized Verification of Compiler Back-Ends* in Proc. Int. Workshop on Software Tools for Technology Transfer (STTT'98), Aalborg, Denmark, July 12–13, 1998.
189. A. Durand. Modeling cache coherence protocol – a case study with FLASH. In U. Glässer and P. Schmitt (eds.), *Proc. 5th Int. Workshop on Abstract State Machines*, pp. 111–126. Magdeburg University, 1998.
- During his research stay in Pisa in 1997/98, upon Börger's suggestion Durand investigated the cache coherence protocol in the Stanford FLASH multiprocessor system, for which he provides a high-level specification and correctness proofs related to data consistency. For a model checking verification of the model using SMV see [425].
190. I. Durdanović. From operational specifications to real architectures. Draft of PhD Thesis (NEC Research Institute Princeton), March 2, 2000.
- This PhD project, supervised by Börger, continues the ideas presented in [171]. An ASM Virtual Architecture is defined as the basis for a comprehensive ASM tool environment. The developed base system contains an ASM parser and a compiler into ASM/VA code which is a form of high-level C++ programs whose actual refinement into C++ is supported by programs in a C++ library.
191. E.A. Community. Name change to replace EA by something better. Electronic Discussion at ea@ira.uka.de, September 6 to October 11 (1996).
- The discussion was proposed with the following motivation: “Algebra” makes the theoreticians think that the approach belongs to the algebraic specification and verification research area – and their dissatisfaction and misjudgement comes from our violating so many (I would say almost all) of their cherished concepts and beliefs. “Algebra” makes the practitioners think that we want them to use algebraic notation and equations or laws – and this is enough for them not even to look further at what we really do. “Evolving” is either not understood at all or in the best of all cases interpreted as implying that the signature should be allowed to change – this comes from the analogy with biological systems where the concept is used that way. (Börger on Sept. 6)
- In a lively discussion, two dozen names were proposed, resulting in Päppinghaus' proposal of (Gurevich's) *Abstract State Machines* to become generally

accepted. Here are the concluding messages of October 10/11 which resume this decision.

From: Erik.Tiden@zfe.siemens.de  
 To: eboerger@prosun.first.gmd.de  
 Subject: Name of the beast.

Dear Prof. Börger, I write in English, so that you can quote me to your community if you wish. The name “Gurevich Machines” is impossible in industry, because it only evokes associations of useless (in industrial practice) theoretical concepts. The name “Abstract State Machines” on the other hand, is fine. That’s also what we will keep on calling them here at Siemens central research. Thus, if you stick to “Gurevich Machines” you will end up with two names. Now, if you regard ASMs as a theoretical exercise, investigation, whatever, into the foundations of CS or some such worthy cause, then you can call them whatever you like of course. If you want to make ASMs into something which is useful in practice, calling them GM is simply foolish. Best regards, Erik Tiden.

Answer of October 11.

From: Egon Börger eboerger@prosun.first.gmd.de  
 To: Erik.Tiden@zfe.siemens.de  
 Subject: Abstract State Machines (Gurevich Machines).

Dear Dr. Tiden, thanks for your valuable comment on the EA name problem which I am going to answer in English so that the whole community can follow this. I do not know whether you did follow the entire discussion; I had started it pushed by the need to find a name which helps those of us who aim at practical (in particular industrial) applications of the specification, verification and code development method which has been built around Gurevich’s notion of evolving algebras. I am glad that through the discussion we have found such a name, namely Gurevich (’s Abstract State) Machines. By the way, the first step to this solution, namely the proposal to call the beasts Abstract State Machines, came from one of your collaborators, Dr. Paepinghaus, to whom I am grateful for his suggestion. Adding the inventor’s name to Abstract State Machines is in accordance with usual practice and provided the chance to conclude the search not with two really different names (EA and ASM) but with ONE name which is generally accepted by the community. Gurevich Machines or Abstract State Machines are not two different names but only shorthands for Gurevich’s Abstract State Machines. Here is another variation, appearing in the title for one of my forthcoming lectures: Abstract State Machines (Gurevich Machines). An interesting feature which makes Gurevich’s ASMs unique is that they have both practical AND theoretical relevance (although surprisingly enough the theoretical potential of the notion of Gurevich Machines has been recognized and explored even less than its practical relevance). Therefore it IS valuable to have a unique name which takes into account the sometimes diverging interests. I hope this is a satisfactory answer to your message. With best wishes, Egon Börger.

192. S. Eilenberg. *Automata, Machines and Languages*, Vol. A. Academic Press, 1974.
193. R. Eschbach. A termination detection algorithm: Specification and verification. In J. Wing, J. C. P. Woodcock, and J. Davies (eds.), *Proc. FM’99, Vol. II*, Lecture Notes in Computer Science, Vol. 1709, pp. 1720–1737. Springer-Verlag, 1999.

A two-level specification of a distributed termination detection algorithm is given using ASMs. The lower-level specification of the algorithm is proved equivalent to the upper-level specification.

194. R. Eschbach, U. Gässer, R. Gotzhein, M. v. Löwis, and A. Prinz. Formal definition of SDL-2000 – compiling and running SDL specifications as ASM models. *J. Universal Computer Science*, 7(11):1025–1050, 2001.  
Contains the most recent and detailed survey of the SDL-2000 formal semantics definition [292] that was accepted in 2000 by ITU-T, the international standardization body for telecommunication. The focus of this survey is on the dynamic semantics, where ASMs have been applied as the underlying framework. In particular, the SDL Abstract Machine (SAM) model including real time, the definition of SAM programs and their execution by the SDL Virtual Machine (SVM) (SDL-to-ASM compiler and further tool support) are presented.
195. R. Eschbach, U. Glässer, R. Gotzhein, and A. Prinz. On the formal semantics of SDL-2000: A compilation approach based on an abstract SDL machine. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 242–265. Springer-Verlag, 2000.  
An overview of the semantics of SDL-2000, whose complete and final definition which uses ASMs appears in [292]. A simplified language SPL is defined and described using ASMs to point out some of the unique features of the semantics of SDL-2000. Also in TIK-Report No. 87, pp. 131–151, ETH Zürich, March 2000.
196. H. Eveking. Machine assisted verification. In E. Börger (ed.), *Architecture Design and Validation Methods*, pp. 191–242. Springer-Verlag, 2000.
197. L. M. G. Feijs and H. B. M. Jonkers. *Formal Specification and Design*. Cambridge University Press, Cambridge, 1992.
198. L. M. G. Feijs, H. B. M. Jonkers, C. P. J. Koymans, and G. R. Renardel de Lavalette. Formal definition of the design language COLD-K. Technical Report No. 234/87, Philips Research Laboratories, 1987.  
Also appeared as ESPRIT Document No. METEOR/t7/PRLE/7, 1987. Final update in August 1989.
199. J. Fitzgerald and P. G. Larsen. *Modeling Systems. Practical Tool and Techniques in Software Development*. Cambridge University Press, Cambridge, 1998.
200. B. Fordham, S. Abiteboul, and Y. Yesha. Evolving databases: An application to electronic commerce. In *Proc. Int. Database Engineering and Applications Sympos. (IDEAS)*, pp. 191–200, Montreal, August 1997.  
The paper describes an ASM-based prototype system, in the spirit of active databases, for specifying electronic commerce applications. An extensible database model called “evolving databases” (EDB) is defined based upon ASMs. It is applied to capture in a rigorously transparent way the state changes involved in electronic commerce negotiations, concerning the traded products, the negotiators, their orders and the laws accepted as the basis for the particular negotiation. See [1].
201. Foundations of Software Engineering Group, Microsoft Research. AsmL. Web pages at <http://research.microsoft.com/foundations/AsmL/>, 2001.
202. G. Franceschinis and M. Ribaud. Efficient performance analysis techniques for stochastic well-formed nets and stochastic-process algebras. In W. Reisig

- and G. Rozenberg (eds.), *Lectures on Petri Nets II: Applications*, Vol. 1492, pp. 386–437. Springer-Verlag, 1998.
203. N. G. Fruja and R. F. Stärk. The hidden computation steps of turbo Abstract State Machines. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 244–262. Springer-Verlag, 2003.
204. N. E. Fuchs. Specifications are (preferably) executable. *Software Eng. J.*, 7(5):323–334, September 1992.  
Reprinted in: J. P. Bowen, M. G. Hinchey, *High-Integrity System Specification and Design*, pp. 583–608. Springer-Verlag, London, 1999.
205. N. E. Fuchs, U. Schwertel, and R. Schwitter. Attempto controlled English – not just another logic specification. In P. Flener (ed.), *Logic-Based Program Synthesis and Transformation, 8th Int. Workshop LOPSTR’98*, Lecture Notes in Computer Science, Vol. 1559, pp. 1–20. Springer-Verlag, 1998.
206. M. Gaieb. *Génération de spécifications Centaur à partir de spécifications Montages*. Master’s thesis, Université de Nice – Sophia Antipolis, France, June 1997.  
This work investigates the possibilities of mapping the operational ASM semantics of the static analysis phase of Montages [311] into the declarative Natural Semantics framework. A formalization for the list arrows of Montages is found – a feature that has not been fully formalized in [311]. In addition, the Gem-Mex Montages tool is interfaced to the Centaur system (which executes Natural Semantics specifications), and the tool support of Centaur is exploited in order to generate structural editors for languages defined with Montages.
207. A. Gargantini and E. Riccobene. Encoding Abstract State Machines in PVS. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 303–322. Springer-Verlag, 2000.  
A framework for automatic translation from ASM to PVS is presented. Following a suggestion by Börger, the ASM specification of the Production Cell problem [120] is used as a case study. Also appears in TIK-Report No. 87, pp. 152–173, ETH Zürich, March 2000.
208. A. Gargantini and E. Riccobene. ASM-based testing: Coverage criteria and automatic test sequence generation. *J. Universal Computer Science*, 7(11):1051–1068, 2001.  
ASMs are used for testing purposes, defining adequacy criteria measuring the coverage achieved by a test suite, and determining whether sufficient testing has been performed. An algorithm is defined to generate from ASMs test sequences with desired coverage, exploiting the counter example generation of SMV. See the continuation in [209].
209. A. Gargantini and E. Riccobene. Using Spin to generate tests from ASM specifications. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 263–277. Springer-Verlag, 2003.  
Continuation of [208].
210. M. C. Gaudel. *Génération et Preuve de Compilateurs Basées sur une Sémantique Formelle des Langages de Programmation*. Thèse, L’Institut National Polytechnique de Lorraine, France, 1980.  
The work to which mostly the idea is attributed of using Tarski structures as the most general notion of states. But see [359].

211. T. Gaul. An Abstract State Machine specification of the DEC-Alpha processor family. Verifix Working Paper Verifix/UKA/4, University of Karlsruhe, Germany, 1995.
- An ASM for the DEC-Alpha processor family is derived directly from the original manufacturer's handbook. The specification omits certain less-used instructions and VAX compatibility parts.
212. T. Gaul, A. Heberle, and W. Zimmermann. An ASM specification of the operational semantics of MIS. Verifix Working Paper Verifix/UKA/3, University of Karlsruhe, Germany, 1998.
- An ASM specification of MIS, an intermediate programming language used in the Verifix project for provably correct compilation to the DEC-Alpha microprocessor [211].
213. A. Gawanmeh, S. Tahar, and K. Winter. Interfacing ASMs with the MDG tool. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 278–292. Springer-Verlag, 2003.
214. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.
- A high-level model of the system described in this book has been developed in [107, 108].
215. S. Gerhart, D. Craigen, and T. Ralston. Experience with formal methods in critical systems. *IEEE Software*, 11(1):21–28, January 1994.
216. F. Giannuzzi. *Studi di un metodo per la derivazione dei casi di test da specifiche ASM*. Tesi di laurea, Università di Pisa, Italy, July 2001.
- Studies the derivation of test cases from ASM specifications, illustrating an application of the cause-effect-graph method for the Production Cell ASM [120]. Supervised by Bertolino and Börger.
217. M. Giese, D. Kempe, and A. Schönege. KIV zur Verifikation von ASM-Spezifikationen am Beispiel der DLX-Pipelining Architektur. Interner Bericht 16/97, Universität Karlsruhe, Germany, 1997.
- The Karlsruhe Interactive Verifier (KIV system) is used for the formal verification of the parallelization of the ASM specification for the serial DLX architecture, the first step of the refinement to its parallel version with five-stage pipelining in [119]. Two additions to the KIV system are described which were designed in the course of this case study.
218. U. Glässer. Systems level specification and modeling of reactive systems: Concepts, methods, and tools. In F. Pichler, R. Moreno-Díaz, and R. Albrecht (eds.), *Computer Aided Systems Theory—EUROCAST'95: Proc. 5th Int. Workshop on Computer Aided Systems Theory* (Innsbruck, Austria, May 1995), Lecture Notes in Computer Science, Vol. 1030, pp. 375–385. Springer-Verlag, 1996.
- The paper investigates the derivation of formal requirements and design specifications at systems level as part of a comprehensive design concept for complex reactive systems. In this context the meaning of correctness with respect to the embedding of mathematical models into the physical world is discussed.
219. U. Glässer. Combining Abstract State Machines with predicate transition nets. In F. Pichler and R. Moreno-Díaz (eds.), *Computer Aided Systems Theory – EUROCAST'97 (Proc. 6th Int. Workshop on Computer Aided Systems*

*Theory, Las Palmas de Gran Canaria, Spain, Feb. 1997*), Lecture Notes in Computer Science, Vol. 1333, pp. 108–122. Springer-Verlag, 1997.

The work investigates the formal relation between ASMs and Pr/TPredicate Transition (Pr/T-) Nets with the aim of integrating both approaches into a common framework for modeling concurrent and reactive system behavior, where Pr/T-nets are considered as a graphical interface for distributed ASMs. For the class of *strict Pr/T-nets* (which constitutes the basic form of Pr/T-nets) a transformation to distributed ASMs is given.

220. U. Glässer. ASM semantics of SDL: Concepts, methods, tools. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz (eds.), *Proc. 1st Workshop of the SDL Forum Society on SDL and MSC*, Informatik-Berichte, Vol. 104 (ISSN 0863-095), pp. 271–280. Humboldt-Universität Berlin, 1998.

Proposal to the SDL Forum to use ASMs for a definition of the semantics of SDL which is abstract but through its operational character is apt to be transformed to an executable model. Detailed in [225].

221. U. Glässer. *Analysis and Validation of Formal Requirement Specifications in Model-Based Engineering of Concurrent Systems*. Habilitationsschrift, University of Paderborn, Germany, 1999.

Contains a systematic treatment of the work started in [225] providing ASM models for the dynamic semantics of SDL. Completed in [292]; see [194] for a survey.

222. U. Glässer, R. Gotzhein, and A. Prinz. Towards a new formal SDL semantics based on Abstract State Machines. In G. v. Bochmann, R. Dssouli, and Y. Lahav (eds.), *SDL'99 – The Next Millenium, Proc. 9th SDL Forum*, pp. 171–190. Elsevier Science B.V., 1999.

Based upon the idea proposed in [225], ASMs are applied to formally define the behavior model of a sample SDL-2000 specification. See also “SDL Formal Semantics Definition” by the same authors, published as University of Paderborn Report No. TR SFBR-99-065, June 1999. See the completion of the work in [292] and the survey [194].

223. U. Glässer, Y. Gurevich, and M. Veanes. An abstract communication model. Technical Report MSR-TR-2002-55, Microsoft Research, Redmond, Washington, May 2002.

From [224] an abstract communication model is extracted.

224. U. Glässer, Y. Gurevich, and M. Veanes. High-level executable specification of the universal plug and play architecture. In *Proc. 35th Hawaii Int. Conf. on System Sciences – 2002*, pp. 1–10. IEEE Computer Society Press, 2002.

An AsmL specification of the Universal Plug and Play (UPnP) architecture for peer-to-peer network connectivity of intelligent devices. A more detailed version appeared in June 2001 as Microsoft Research Technical Report MSR-TR-2001-59 under the title “Universal Plug and Play Models”. See [223].

225. U. Glässer and R. Karges. Abstract State Machine Semantics of SDL. *J. Universal Computer Science*, 3(12):1382–1414, 1997.

A formal semantic model of Basic SDL-92 – according to the *ITU-T Recommendation Z.100* – is defined in terms of an abstract SDL machine based on the concept of a *multi-agent real-time ASM*. The resulting interpretation model is not only mathematically precise but also reflects the common understanding of SDL in a direct and intuitive manner; it provides a *concise* and *understandable* representation of the complete dynamic semantics of Basic SDL-92. Moreover, the model can easily be *extended* and *modified*. The

article considers the behavior of channels, processes and timers with respect to signal transfer operations and timer operations. Continuation of this work and merging it with work by Gotzhein and by Prinz [222, 221, 367] led to the ITU-T standard definition of SDL-2000 [292, 195].

226. U. Glässer and P. Schmitt (eds.). *Proc. 5th Int. Workshop on Abstract State Machines*, Germany, 1998. GI Jahrestagung 1998, Otto-von-Guericke-Universität Magdeburg.

Extended abstracts of the talks presented to the workshop which was organized as part of the 28th Annual Conf. of the German Computer Science Society (GI Jahrestagung). See [435, 395, 328, 188, 274, 414, 189, 46, 169].

227. P. Glavan and D. Rosenzweig. Communicating evolving algebras. In E. Börger, H. Kleine Büning, G. Jäger, S. Martini, and M. M. Richter (eds.), *Computer Science Logic*, Lecture Notes in Computer Science, Vol. 702, pp. 182–215. Springer-Verlag, 1993.

A theory of concurrent computation within the framework of ASMs is developed, generalizing [257, 122]. As an illustration models are given for the Chemical Abstract Machine and the  $\pi$ -calculus. See [248] for a more general definition of the notion of distributed ASM runs.

228. P. Glavan and D. Rosenzweig. Evolving algebra model of programming language semantics. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 416–422, Elsevier, Amsterdam, 1994.

Defines an ASM interpretation of many-step SOS, denotational semantics and Hoare logic for the language of while-programs and states correctness and completeness theorems, based on a simple flowchart model of the language.

229. W. Goerigk, A. Dold, T. Gaul, G. Goos, A. Heberle, F. W. von Henke, U. Hoffmann, H. Langmaack, H. Pfeifer, H. Ruess, and W. Zimmermann. Compiler correctness and implementation verification: The verifix approach. In P. Fritzsion (ed.), *Int. Conf. on Compiler Construction, Proc. Poster Session of CC'96*, Linköping, Sweden, 1996. IDA Technical Report LiTH-IDA-R-96-12.

In this project a method is developed to establish, modulo hardware correctness, the correctness of reliable initial compilers (not only compiler specifications) for an appropriate high-level system programming language. The approach is based upon multiple-phase compilation (with closely related intermediate languages) and a diagonal bootstrapping technique. The following three major steps are performed. (1) Verification of a specification of the compilation function with respect to the semantics of the source and target language and a correctness definition. Here ASMs are used to rigorously define source and target language semantics and the correctness property. PVS is used for proof support. (2) Verification of a compiler implementation in a high-level language, using generators to generate the front end and parts of the back end. Small (proven to be correctly implemented) checker routines are used to verify by syntactical a posteriori code inspection that the input and output of the generators have the needed properties (program checking). (3) Verification of a compiler implementation in binary. An initial bootstrap compiler is used which is proved (once) to be correctly implemented in binary. In the project this is a compiler from COMLISP to Transputer code, whose semantics are defined by SOS methods. No further binary code verification is necessary. For the program checker and other system software it suffices to implement them correctly in the high-level source language of the initial compiler (using standard program transformation or verification techniques).

230. G. Goos, A. Heberle, W. Löwe, and W. Zimmermann. On modular definitions and implementations of programming languages. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines – ASM 2000, Int. Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, TIK-Report, No. 87, pp. 174–208. ETH Zürich, March 2000.
- A formal composition and refinement (correct implementation) mechanism for state-transition systems is presented which exploits the abstract syntax of programs. Applications are made to language semantic definitions using ASMs. Montages [311] is characterized as a set of parameterized ASMs.
231. G. Goos and W. Zimmermann. Verifying compilers and ASMs. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 177–202. Springer-Verlag, 2000.
- ASMs are used to describe verifying compilers: compilers which verify the correctness of their generated code.
232. G. Gottlob, G. Kappel, and M. Schrefl. Semantics of object-oriented data models – the evolving algebra approach. In J. W. Schmidt and A. A. Stogny (eds.), *Next Generation Information Technology*, Lecture Notes in Computer Science, Vol. 504, pp. 144–160. Springer-Verlag, 1991.
- Uses ASMs to define, in the context of a graphical object-oriented data model design language, the operational semantics of object creation, of overriding and dynamic binding, and of inheritance at the type level (type specialization) and at the instance level (object specialization). Issued also as technical report MooD-TR 90/02, Technische Universität Wien, December 20, 1990. See [397].
233. E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998.
- Computer systems, *e.g.* databases, are not necessarily finite because they may involve, for example, arithmetic. Motivated by such computer science challenges and by ASM applications, metafinite structures, as they typically appear in ASM states, are defined and finite model theory is extended to metafinite models. An early version has been presented under the title *Towards a Model Theory of Metafinite Structures* to the Logic Colloquium 1994; see the abstract in the *J. Symbolic Logic*. An intermediate version appeared in *Logic and Computational Complexity, Selected Papers*, Lecture Notes in Computer Science, Vol. 960, pp. 313–366, Springer-Verlag, 1995.
234. E. Grädel and A. Nowack. Quantum computing and Abstract State Machines. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003–Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 309–323. Springer-Verlag, 2003.
- Derives the ASM thesis for quantum algorithms from postulates which are inspired by the axiomatization of parallel algorithms in [61].
235. E. Grädel and M. Spielmann. Logspace reducibility via Abstract State Machines. In J. Wing, J. Woodcock, and J. Davies (eds.), *Proc. FM’99, Vol. II*, Lecture Notes in Computer Science, Vol. 1709, pp. 1738–1757. Springer-Verlag, 1999.
- ASMs are used to investigate logspace reducibility among structures, capturing the choiceless fragment of logspace. A continuation of [62]. See also [402].
236. I. Graham. *The Transputer Handbook*. Prentice-Hall, 1990.

- Together with [289, 290], this book served as the basis for the ASM model developed for the Transputer in [104].
237. W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from Abstract State Machines. In *Software Engineering Notes*, Vol. 27.4, pp. 112–122. ISSTA 02, 2002.
- Proposes a scheme for grouping ASM states into finitely many equivalence classes (“hyperstates”) on which an FSM is induced to reflect the ASM transitions for testing purposes. A preliminary version was presented to the Int. ASM’01 Workshop in February 2001 [110]. It appeared in October 2001 under the title “Conformance Testing with Abstract State Machines” as MSR-TR-2001-97 and in May 2002 revised under the same number with the new title.
238. R. Groenboom and G. R. Renardel de Lavalette. A formalization of evolving algebras. In S. Fischer and M. Trautwein (eds.), *Proc. Accolade 95*, pp. 17–28. Dutch Research School in Logic, ISBN 90-74795-31-5, 1995.
- The authors present the syntax and semantics for a Formal Language for Evolving Algebra (FLEA) covering sequential ASMs. This language is then extended to a multi-modal language FLEA’, and it is sketched how one can transfer the axioms of the logic MLCM to FLEA’. MLCM is a Modal Logic of Creation and Modification, a dynamic logic which is incorporated in Jonker’s Common Object-Oriented Language for Design, COLD [197, 198]. See [405].
239. M. Grosse-Rhode. A formal specification framework for evolving algebras. Unpublished manuscript. Technical University of Berlin, 1996.
- Applies some algebraic-categorical composition schemes to ASMs, illustrated on an alternating-bit protocol specification.
240. Y. Gurevich. Reconsidering Turing’s Thesis: Toward more realistic semantics of programs. Technical Report CRL-TR-36-84, EECS Department, University of Michigan, September 1984.
- An attempt to reconsider Turing’s Thesis, taking into account that resources are bounded. The earliest known paper in which the ideas behind ASMs began to take form. See the continuation in [241].
241. Y. Gurevich. A new thesis. *Abstracts, American Mathematical Society*, 6(4):317, August 1985.
- Following [240], for the first time the ASM Thesis is stated, but no definition for ASMs is given yet. See the continuation in [244, 242].
242. Y. Gurevich. Algorithms in the world of bounded resource. In R. Herken (ed.), *The Universal Turing Machine – A Half-Century Story*, pp. 407–416. Oxford University Press, 1988.
- Early complexity theoretical motivation for the introduction of ASMs is discussed.
243. Y. Gurevich. Kolmogorov machines and related issues. *Bull. EATCS*, 35:71–82, 1988.
- The Kolmogorov–Uspenskii thesis is stated that every computation, performing only one restricted local action at a time, can be viewed as the computation of an appropriate Komogorov–Uspenskii machine.
244. Y. Gurevich. Logic and the challenge of computer science. In E. Börger (ed.), *Current Trends in Theoretical Computer Science*, pp. 1–57. Computer Science Press, 1988.
- Part 2 contains the first small examples for ASMs, drawn from Gurevich’s lectures in Semantics of Programming Languages delivered in Pisa in May 1986 (not “in the Spring of 1987” as stated erroneously, e.g. in [92]).

245. Y. Gurevich. Evolving algebras. A tutorial introduction. *Bull. EATCS*, 43:264–284, 1991.
- The ASM thesis is stated. A slightly revised version was reprinted under the title “Evolving Algebras: An attempt to discover semantics” in G. Rozenberg and A. Salomaa Eds, *Current Trends in Theoretical Computer Science*, World Scientific, 1993, pp. 266–292. A german textbook version of the definition appeared in [73]. For a more elaborate and complete definition see [248].
246. Y. Gurevich. Evolving Algebras. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 423–427, Elsevier, Amsterdam, 1994.
- The opening talk at the first ASM workshop. Sections: Introduction, The ASM Thesis, Remarks, Future Work.
247. Y. Gurevich. Logic activities in Europe. *ACM SIGACT News*, 25(2):11–24, 1994.
- A critical analysis of European logic activities in computer science. Subsection 4.6 *Mathematics and Pedantics* discusses the separation of different levels of verification in the context of modeling with ASMs.
248. Y. Gurevich. Evolving algebras 1993: Lipari Guide. In E. Börger (ed.), *Specification and Validation Methods*, pp. 9–36. Oxford University Press, 1995.
- The notion of sequential ASMs defined in [245] is extended to cover distributed computations. A later update *May 1997 Draft of the ASM Guide* appeared as Technical Report CSE-TR-336-97, EECS Dept., University of Michigan.
249. Y. Gurevich. Sequential Abstract State Machines capture sequential algorithms. *ACM Trans. Computational Logic*, 1(1):77–111, July 2000.
- The notion of “sequential algorithm” is axiomatized to derive from three basic axioms the sequential version of the “ASM thesis” which was proposed in [241, 245]. An early version appeared under different titles as Microsoft Research Technical Reports MSR-TR-99-09 and MSR-TR-99-65, and in Bull. EATCS 67 (February 1999), 93–124. See [61] for an extension to the notion of “synchronous parallel algorithms”.
250. Y. Gurevich and E. Börger. Evolving algebras – mini course. BRICS Technical Report BRICS-NS-95-4, ISSN 0909-3206, University of Aarhus, Denmark, July 1995.
- Contains reprints of the papers [56, 245, 246, 248, 252, 254, 251, 114, 102, 104, 108] which were used as material for a course on ASMs delivered by the two authors at BRICS, Aarhus, in the summer of 1995.
251. Y. Gurevich and J. Huggins. The semantics of the C programming language. In E. Börger, H. Kleine Büning, G. Jäger, S. Martini, and M. M. Richter (eds.), *Computer Science Logic*, Lecture Notes in Computer Science, Vol. 702, pp. 274–309. Springer-Verlag, 1993.
- The method of successive refinements is used to give a succinct dynamic semantics of the C programming language. For a correction of minor errors and omissions see the ERRATA in Lecture Notes in Computer Science, Vol. 832, pp. 334–336, Springer-Verlag, 1994. An early version appeared under the title *The Evolving Algebra Semantics of C: Preliminary Version* as Technical Report CSE-TR-141-92, EECS Department, University of Michigan, Ann Arbor, 1992. This work is included in the PhD thesis *Evolving Algebras: Tools for Specification, Verification, and Program Transformation* of the second author, pp. IX+91, supervised by Gurevich at the University of Michigan, Ann

- Arbor, 1995. For an extension to C++ see [420]. For an addition of the statics of C to the model see [285].
252. Y. Gurevich and J. Huggins. Evolving algebras and partial evaluation. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 587–592, Elsevier, Amsterdam, 1994.
- The paper describes an automated partial evaluator for sequential ASMs implemented at the University of Michigan. It takes an ASM and a portion of its input and produces a specialized ASM using the provided input to execute rules when possible and generating new rules otherwise. A full version appears as J. Huggins, “An Offline Partial Evaluator for Evolving Algebras”, Technical Report CSE-TR-229-95, EECS Department, University of Michigan, Ann Arbor, 1995. This work is included in the PhD thesis *Evolving Algebras: Tools for Specification, Verification, and Program Transformation* of the second author, pp. IX+91, University of Michigan, Ann Arbor, 1995. For an extension of this work see [178].
253. Y. Gurevich and J. Huggins. The railroad crossing problem: An experiment with instantaneous actions and immediate reactions. In *Proc. CSL’95 (Computer Science Logic)*, Lecture Notes in Computer Science, Vol. 1092, pp. 266–290. Springer-Verlag, 1996.
- An ASM solution for the railroad crossing problem in [277]. The paper experiments with agents that perform instantaneous actions in continuous time at the moment they are enabled. A preliminary version appeared under the title *The Railroad Crossing Problem: An Evolving Algebra Solution* as research report LITP 95/63 of Centre National de la Recherche Scientifique, Paris, and under the title *The Generalized Railroad Crossing Problem: An Evolving Algebra Based Solution* as research report CSE-TR-230-95 of EECS Department, University of Michigan, Ann Arbor, MI. For a further investigation see [34, 35].
254. Y. Gurevich and J. Huggins. Equivalence is in the eye of the beholder. *Theoretical Computer Science*, 179(1–2):353–380, 1997.
- A response to a paper of Leslie Lamport, “Processes are in the Eye of the Beholder” which is published in the same volume. It is discussed how the same two algorithms may and may not be considered equivalent. In addition, a direct proof is given of an appropriate equivalence of two particular algorithms considered by Lamport. A preliminary version appeared as research report CSE-TR-240-95, EECS Dept., University of Michigan, Ann Arbor, Michigan 1995.
255. Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.). *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912. Springer-Verlag, 2000.
- Proc. Int. Workshop ASM2000 held at Monte Verità, Switzerland, March 2000.
256. Y. Gurevich and R. Mani. Group membership protocol: Specification and verification. In E. Börger (ed.), *Specification and Validation Methods*, pp. 295–328. Oxford University Press, 1995.
- A processor-group membership protocol involving timing constraints is formally specified and verified using distributed ASMs.
257. Y. Gurevich and L. S. Moss. Algebraic operational semantics and Occam. In E. Börger, H. Kleine Büning, and M. M. Richter (eds.), *CSL’89, 3rd Workshop on Computer Science Logic*, Lecture Notes in Computer Science, Vol. 440, pp. 176–192. Springer-Verlag, 1990.

The first application of ASMs to distributed parallel computing with the challenge of true concurrency. For an improved (no longer parse tree determined, but truly concurrent) ASM model for Occam and its refinement to a Transputer implementation see [105, 104].

258. Y. Gurevich and D. Rosenzweig. Partially ordered runs: A case study. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 131–150. Springer-Verlag, 2000.

The ASM investigation [114] of Lamport’s Bakery Algorithm is sharpened in terms of partially ordered runs, abstracting from the mapping of moves to linear realtime. Some properties are proved which are useful for reasoning about partially ordered runs. The paper also appeared as a technical report in TIK-Report No. 87, ETH Zürich, March 2000, and in MSR-TR-99-98.

259. Y. Gurevich, W. Schulte, and C. Wallace. Investigating Java concurrency using Abstract State Machines. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 151–176. Springer-Verlag, 2000.

An ASM specification and verification of Java’s model of concurrency, including threads and synchronization. Also in TIK-Report No. 87, pp. 227–271, ETH Zürich, March 2000, and in University of Delaware Department of Computer & Information Sciences TR 2000-04.

260. Y. Gurevich, N. Soparkar, and C. Wallace. Formalizing database recovery. *J. Universal Computer Science*, 3(4):320–340, 1997.

A database recovery algorithm (the undo-redo algorithm) is modeled at several levels of abstraction, with verification of the correctness of the high-level model and of each of the four refinement steps. An updated version of the Technical Reports CSE-TR-249-95 and CSE-TR-327-97 of EECS Department, University of Michigan, Ann Arbor, and of the paper *Formalizing Recovery in Transaction-Oriented Database Systems* of C. Wallace and Y. Gurevich and N. Soparkar, published in S. Chaudhuri and A. Deshpande and R. Krishnamurthy (eds.): Proc. 7th Int. Conf. on Management of Data, Tata McGraw-Hill, New Delhi, India, 1995, pp. 166–185.

261. Y. Gurevich and M. Spielmann. Recursive Abstract State Machines. *J. Universal Computer Science*, 3(4):233–246, 1997.

A definition of recursive ASMs in terms of distributed ASMs is suggested. A preliminary version appeared as Technical Report CSE-TR-322-96, EECS Department, University of Michigan, Ann Arbor, 1996. For a definition of recursive ASMs in terms of sequential ASMs see [134].

262. Y. Gurevich and N. Tillmann. Partial updates: Exploration. *J. Universal Computer Science*, 7(11):918–952, 2001.

A solution is proposed for the problem of cumulative updates for counters, steps and maps. See the continuation in [263].

263. Y. Gurevich and N. Tillmann. Partial updates exploration II. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 57–86. Springer-Verlag, 2003.

Continuation of [262].

264. Y. Gurevich and C. Wallace. Specification and verification of the Windows Card runtime environment using Abstract State Machines. Technical Report MSR-TR-99-07, Microsoft Research, Redmond, Washington, February 1999.

- An ASM specification of the Windows Card Runtime Environment and a verification of certain safety properties.
265. J. V. Guttag, E. Horowitz, and D. R. Musser. Abstract data types and software validation. *Commun. ACM*, 21(12), 1978.
  266. N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.
  267. J. A. Hall. Taking Z seriously. In *ZUM'97*, Lecture Notes in Computer Science, Vol. 1212, pp. 89–91. Springer-Verlag, 1997.
  268. D. Harel. Dynamic logic. In D. M. Gabbay and F. Guenther (eds.), *Handbook of Philosophical Logic*, Vol. II, pp. 497–604. Reidel, Dordrecht, 1983.
  269. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
  270. D. Harel and R. Marelly. Capturing and executing behavioral requirements: the play-in/play-out approach. Technical Report MCS01-15, Weizmann Institute of Science, Israel, 2001.
  271. D. Harel and M. Politi. *Modeling reactive systems with statecharts*. McGraw-Hill, 1998.
  272. P. Hartel and L. Moreau. Formalizing the safety of Java, the Java Virtual Machine and Java Card. *ACM Computing Surveys*, 33(4):517–558, 2001.  
A review of the literature on formal approaches of Java and its implementation with focus on safety issues and their impact on smart cards. Sect. 6.2 evaluates the ASM based work in this area [138, 137, 139, 140, 141, 406, 421].
  273. A. Heberle. *Korrekte Transformationsphase – der Kern korrekter Übersetzer*. PhD thesis, Universität Karlsruhe, Germany, 2000.  
The essential results of the thesis (which is written in German) are published in [274, 275].
  274. A. Heberle and W. Löwe. On ASM-based specification of programming language semantics and reusable correct compilations. In U. Glässer and P. Schmitt (eds.), *Proc. 5th Int. Workshop on Abstract State Machines*, pp. 68–90. Magdeburg University, 1998.  
General equivalence-preserving transformations on ASM specifications of programming languages are defined, to be used for the definition of provably correct compilation schemes. An extensible language AL is introduced for specifying dynamic language semantics in a way which facilitates the reuse of verified transformations. Some of the results are from [273].
  275. A. Heberle, W. Löwe, and M. Trapp. Safe reuse of source to intermediate language compilations. In R. Chillarege (ed.), *Proc. 9th. Int. Symp. on Software Reliability Engineering*, Fast Abstract and Industrial Tracts, Paderborn, Germany, 4–7 November 1998.  
See <http://www.chillarege.com/issre/fastabstracts/98417.html>.  
Contains some results of [273].
  276. C. Heitmeyer. Using SCR methods to capture, document, and verify computer system requirements. In E. Börger, B. Hörger, D. L. Parnas, and D. Rombach (eds.), *Requirements Capture, Documentation, and Validation*. Dagstuhl Seminar No. 99241, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, 1999.
  277. C. Heitmeyer and D. Mandrioli. *Formal Methods for Real-Time Computing*, Trends in Software, Vol. 5. John Wiley, 1996.

- Extensive study of the Railroad Crossing Problem, proposed as a case study for real-time computing and solved using various popular specification and verification methods. For an ASM solution see [253].
278. J. L. Hennessy and D. A. Patterson. *Architecture: A Quantitative Approach*. Morgan Kaufman, 2nd edn., 1996.
279. H. Hinrichsen. Formally correct construction of a pipelined DLX architecture. Technical Report TR 98-5-1, Darmstadt University of Technology, Dept. of Electrical and Computer Engineering, Germany, 1998.
- In an e-mail to Börger on February 11, 1998, Hinrichsen points out that for a correct handling of the instruction sequence 1. LOAD R1 A, 2. LOAD R2 B, 3. ADD R3 R1 R2, the ADD instruction must be stalled for one clock cycle. This corrects an omission of a hazard case in the last refinement step of [119].
280. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 583, 1969.
281. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
282. M. Holcombe and F. Ipate. *Correct Systems. Building a Business Process Solution*. Springer-Verlag, 1998.
283. J. Huggins. Kermit: Specification and verification. In E. Börger (ed.), *Specification and Validation Methods*, pp. 247–293. Oxford University Press, 1995.
- The Kermit file-transfer protocol [163] is specified and verified using ASMs at several layers of abstraction. This work is part of the author’s PhD thesis *Evolving Algebras: Tools for Specification, Verification, and Program Transformation*, pp. IX+91, University of Michigan, Ann Arbor, 1995.
284. J. Huggins. Broy-Lampert Specification Problem: A Gurevich Abstract State Machine Solution. Technical Report CSE-TR-320-96, EECS Dept., University of Michigan, 1996.
- Upon Börger’s suggestion, Huggins developed an ASM solution to the specification problem proposed by Broy and Lampert, in conjunction with the Dagstuhl Seminar on Reactive Systems, held in Dagstuhl, Germany, 26–30 September, 1994. A preliminary version appeared as Technical Report CSE-TR-223-94, EECS Department, University of Michigan, Ann Arbor, 1994. Other solutions of this problem were published in [144].
285. J. Huggins and W. Shen. The static and dynamic semantics of C. Technical Report CPSC-2000-4, Kettering University, Computer Science Program, Flint, Michigan, 2000.
- The ASM for C in [251] is extended to provide both static and dynamic semantics for C, using Montages [311]. An extended abstract appears in Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, eds., *Abstract State Machines – ASM 2000, Int. Workshop on Abstract State Machines*, Monte Verita, Switzerland, Local Proceedings, TIK-Report No. 87, pp. 272–283, ETH Zürich, March 2000. A previous version appears as Kettering University Computer Science Program Technical Report CPSC-1999-1.
286. J. Huggins and D. Van Campenhout. Specification and verification of pipelining in the ARM2 RISC microprocessor. *ACM Trans. Des. Autom. of Electron. Syst.*, 3(4):563–580, 1998.
- An extended abstract describing a layered ASM specification of the advanced RISC machine processor ARM2, one of the early commercial RISC microprocessors. The method developed in [119] is applied for the layered specification and the correctness proof for the ARM2’s pipelining techniques. In

- [412] this ASM model of the ARM is used to illustrate an approach to automatically transform register transfer descriptions of microprocessors into executable ASMs. A full version of the paper appears as University of Michigan EECS Department Technical Report CSE-TR-371-98. An earlier version appears in *Proc. IEEE Int. High Level Design Validation and Test Workshop (HLDTV'97)*, November 1997.
287. M. Ibanez and H. Rempp. European user survey analysis. Technical Report TR 95104, European Software Institute, Bilbao, Spain, January 30 1996.
288. IEEE Std 1076-1993. *IEEE Standard VHDL Language Reference Manual*. IEEE, New York, USA, 1993.  
The standard description of the hardware design language VHDL'93 which has been formalized by an ASM ground model in [111, 112].
289. INMOS. *Transputer Instruction Set – A Compiler Writer's Guide*. Prentice-Hall, Englewood Cliffs, NJ, 1988.  
INMOS Document 72 TRN 119 05. See the comment to [236].
290. INMOS. *Transputer Implementation of Occam – Communication Process Architecture*. Prentice-Hall, Englewood Cliffs, NJ, 1989.  
See comment to [236].
291. ISO. Prolog-Part 1: General core. ISO Standard Information Technology–Programming Languages ISO/IEC 13211-1, ISO/ICE, January 1995.
292. ITU-T. SDL formal semantics definition. ITU-T Recommendation Z.100 Annex F, International Telecommunication Union, November 2000.  
This document contains the complete, internationally standardized formal semantics definition of SDL-2000, a design language for the development of distributed real-time systems in general and telecommunication systems in particular. SDL is industrially applied in the telecommunications industry, for instance, to the development of UMTS protocols and Intelligent Networks. The dynamic semantics of SDL-2000 is defined using ASMs as the underlying mathematical framework. For further information see <http://rn.informatik.uni-kl.de/projects/sdl>. For a survey see [194].
293. J. W. Janneck. *Syntax and Semantics of Graphs*. PhD thesis, ETH Zürich, Switzerland, 2000.  
Published in: *Berichte aus der Informatik, TIK Series Vol. 38*, Shaker Verlag Aachen (ISBN 3-8265-7688-8), pp. XI+177. The classical networks of stream processing finite state machines (with their notion of network components with input and output ports to communicate among each other) are enriched by ASM state transformations of individual components. The resulting machines are applied to give a uniform rigorous semantics to common visual notations for discrete event systems, together with a prototypical implementation. Illustration by Petri nets.
294. J. W. Janneck and P. Kutter. Mapping automata: Simple Abstract State Machines. TIK-Report 49, ETH Zürich, Switzerland, June 1998.  
*Mapping automata* are defined as ASMs where the state is formed by a single binary function (interpreted as mapping which assigns to every object in the base set  $U$  a unary function over objects), and the rules are built from updates of that binary function in the usual way. Using the standard coding of arbitrary structures into the structure of one binary function, the resulting correspondence between mapping automata and ASMs is shown to preserve the desired computational equivalence. Also appears in Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, eds., *Abstract State Machines – ASM 2000*,

- Int. Workshop on Abstract State Machines*, Monte Verita, Switzerland, Local Proceedings, TIK-Report No. 87, pp. 310–325, ETH Zürich, March 2000. An implementation in Java is reported in *Object-Based Mapping Automata (Reference Manual)* by J. W. Janneck, TIK-Report No. 50, ETH Zürich, June 1998. Mapping automata are used in [296] for a description of the semantics of UML statecharts.
295. J. W. Janneck and P. Kutter. Object-based Abstract State Machines. TIK-Report 47, ETH Zürich, Switzerland, 1998.
- Proposes to view ASMs as classes attached to objects which communicate only by message passing. Illustration by a class definition for Petri net places and transitions.
296. Y. Jin, R. Esser, and J. W. Janneck. Describing the syntax and semantics of UML statecharts in a heterogeneous modeling environment. In *Proc. DIA-GRAMS 2002*, pp. 320–334, 2002.
- Based upon the syntactical description of UML statecharts by attributed graphs coming with well-formedness conditions, the mapping automata of [294] are used to describe the semantics of UML statecharts. Compared with the ASM model of UML statecharts in [99], the focus here is on a discussion of transition conflicts.
297. D. E. Johnson and L. S. Moss. Grammar formalisms viewed as Evolving Algebras. *Linguistics and Philosophy*, 17:537–560, 1994.
- Distributed ASMs are used to model formalisms for natural language syntax. The authors start by defining an ASM model of context-free derivations which abstracts from the parse tree descriptions used in [257, 123] and from the dynamic tree generation appearing in [127, 131]. Then the basic model of context-free rules is extended to characterize in a uniform and natural way different context-sensitive languages in terms of ASMs. See [341, 342].
298. J. Jürjens. *Principles for secure system development*. PhD thesis, Wolfson College Oxford, England, 2001.
299. A. Kalinov, A. Kossatchev, A. Petrenko, M. Posypkin, and V. Shishkov. Using ASM specifications for compiler testing. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, p. 415. Springer-Verlag, 2003.
- Also presented under the title “Using ASM specification for automatic test suite generation for mpC parallel programming language compiler” to AS2000 (4th international workshop on Action Semantics and related frameworks), Copenhagen, July 2002.
300. A. Kaplan and J. Wileden. Formalization and application of a unifying model for name management. In *Proc. 3rd ACM SIGSOFT Sympos. on the Foundations of Software Engineering*, Software Engineering Notes, Vol. 20(4), pp. 161–172, October 1995.
- Presents a unifying model for name management, using ASMs as the specification language for the model. A preliminary version appeared in July 1995 as CMPSCI Technical Report 95-60 of Computer Science Department, University of Massachusetts, Amherst.
301. A. M. Kappel. *Implementation of Dynamic Algebras with an Application to Prolog*. Diplom thesis, Computer Science Dept., Universität Dortmund, Germany, 1990.

This Diplom thesis was triggered by Börger’s lectures on ASM models for Prolog [71, 72], delivered in June 1989 to A. B. C. Cremers’ and H. Ganzinger’s “Diplomanden-und Doktorandenseminar” at the University of Dortmund. Kappel defines a language for the specification of sequential ASMs and designs an abstract target machine (namely a Prolog program) for executing a class of sequential ASMs, including those of the ASM models for Prolog in [71, 72]. A prototype of the compiler has been implemented in Prolog, all the examples have been tested for Quintus Prolog on a SPARC station 1+ and for LPA Prolog on an IBM PC AT. A short version of the paper appeared in [302]; a parallel extension of the interpreter appears in [148].

302. A. M. Kappel. Executable specifications based on dynamic algebras. In A. Voronkov (ed.), *Logic Programming and Automated Reasoning*, Lecture Notes in Artificial Intelligence, Vol. 698, pp. 229–240. Springer-Verlag, 1993. Short version of [301].
303. C. Kern and M. Greenstreet. Formal verification in hardware design: A survey. *ACM Trans. Des. Autom. of Electron. Syst.*, 4:123–193, 1999.
304. C. M. R. Kintala, Kong-Yee Pun, and D. Wotschke. Concise representations of regular languages by degree and probabilistic finite automata. *Math. Systems Theory*, 26(4):379–395, 1993.
305. E. Kohlbrenner, D. Morris, and B. Morris. The history of virtual machines. Web pages at <http://cne.gmu.edu/itcore/virtualmachine/history.htm>.
306. A. N. Kolmogorov and V. A. Uspenskii. On the definition of an algorithm. *AMS Translations, 2nd Series*, 29:217–245, 1993.
307. T. Kropf. *Introduction to Formal Hardware Verification*. Springer-Verlag, 1999.
308. P. Kutter. An ASM macro language for sets. TIK-Report 34, ETH Zürich, Switzerland, January 1998.  
A small set of simple, generic macros that allow one to manipulate and parametrize sets in ASMs, without changing the semantics given in [248].
309. P. Kutter. *Montages – Engineering of Computer Languages*. PhD thesis, ETH Zürich, Switzerland, 2002.  
Contains a denotational semantics of XASM [15] (announced on June 5, 2002 as TIK Report No. 136 under the title “The formal definition of Anlauff’s eXtensible Abstract State Machines”), which is an ASM semantics of Montages, an example language illustrating the description of language features found in sequential Java (see [421]).
310. P. Kutter and A. Pierantonio. The formal specification of Oberon. *J. Universal Computer Science*, 3(5):443–503, 1997.  
A presentation of the syntax, static semantics and dynamic semantics of Oberon, using ASMs and Montages [311]. The dynamic semantics previously appeared as P. Kutter, “Dynamic Semantics of the Oberon Programming Language”, TIK-Report No. 25, ETH Zürich, February 1997.
311. P. Kutter and A. Pierantonio. Montages: Specifications of realistic programming languages. *J. Universal Computer Science*, 3(5):416–442, 1997.  
The authors introduce Montages, a version of ASMs specifically tailored for specifying the static and dynamic semantics of programming languages. Montages combine graphical and textual elements to yield specifications similar in structure, length and complexity to those in common language manuals, but with a formal semantics. A preliminary version appeared in July 1996 under

the title *Montages: Unified Static and Dynamic Semantics of Programming Languages* as Technical Report 118 of Università de L'Aquila. At that same university also the first application of Montages appeared in a Tesi di Laurea [181]. See [19] for an extension of Montages with a finite-state machine model.

312. P. Kutter, D. Schweizer, and L. Thiele. Integrating domain specific language design in the software life cycle. In *Proc. Int. Workshop on Current Trends in Applied Formal Methods*, Lecture Notes in Computer Science, Vol. 1641, pp. 196–212. Springer-Verlag, 1998.

A report on an industrial case study, applying ASMs and Montages [311] to the design, specification and implementation of a driver specification language needed in the context of a complex data warehouse problem at Union Bank of Switzerland.

313. K. Kwon. A structured presentation of a closure-based compilation method for a scoping notion in logic programming. *J. Universal Computer Science*, 3(4):341–376, 1997.

An extension to logic programming which permits scoping of procedure definitions is described at a high level of abstraction (using ASMs) and refined (in a provably-correct manner) to a lower level, building upon the method developed in [132]. The PhD thesis upon which this paper is based was submitted to Duke University on December 12, 1994, under the title “Towards a Verified Abstract Machine for a Logic Programming Language with a Notion of Scope”, No. CS 1994-36, pp. 189.

314. L. Lamport. A new solution of Dijkstra’s concurrent programming problem. *Commun. ACM*, 17(8):453–455, 1974.

Definition of the bakery algorithm to solve the mutual exclusion problem; see also [315]. An ASM analysis of this algorithm appears in [114].

315. L. Lamport. On interprocess communication. Part I: Basic formalism. Part II: Algorithms. *Distributed Computing*, 1:77–101, 1986.

See [314].

316. B. W. Lampson. Principles of computer systems. MIT Lecture Notes 6.826 and <http://research.microsoft.com/Lampson>, Spring 1999.

317. P. J. Landin. A  $\lambda$ -calculus approach. In L. Fox (ed.), *Advances in Programming and Non-Numerical Computation*, pp. 97–141. Pergamon Press London, 1966.

See also the paper *A formal description of Algol 60* by the same author in T. B. Steel (ed.), *Formal Language Description Languages for Computer Programming*, North-Holland, Amsterdam, 1966.

318. H. Langmaack. The ProCoS approach to correct systems. *Real-Time Systems*, 13:253–275, 1997.

319. L. Lavagno, A. Sangiovanni-Vincentelli, and E. M. Sentovitch. Models of computation for system design. In E. Börger (ed.), *Architecture Design and Validation Methods*, pp. 243–295. Springer-Verlag, 2000.

320. N. G. Leveson. Completeness in formal specification language design for process-control systems. In *Formal Methods in Software Practice*, pp. 75–87. ACM Press, 2000.

321. N. G. Leveson and J. D. Reese. SpecTRM: A toolset to support the safeaware methodology. In *Proc. 16th Int. System Safety Conf.*, pp. 256–262, 1998.

322. N. G. Leveson, J. D. Reese, S. Koga, L. D. Pinnel, and S. D. Sandys. Analyzing requirements specifications for mode confusion errors. In *Proc. Workshop on Human Error and System Development*, Glasgow, Scotland, 20–22 March 1997.
323. T. Lindner. Task description. In C. Lewerentz and T. Lindner (eds.), *Formal Development of Reactive Systems. Case Study “Production Cell”*, Lecture Notes in Computer Science, Vol. 891, pp. 9–21. Springer-Verlag, 1995.  
Description of the Production Cell case study which has been derived from a metal-processing plant in Karlsruhe. The book contains solutions of the problem which use various formal methods. The book inspired work on an ASM solution of the problem; see [120].
324. A. Lisitsa and G. Osipov. Evolving algebras and labelled deductive systems for the semantic network based reasoning. In *Proc. Workshop on Applied Semiotics, ECAI’96*, pp. 5–12, August 1996.  
ASMs are used to present the high-level semantics for MIR, an AI semantic network system. Another formalization of MIR is given in terms of labeled deduction systems, and the two formalizations are compared.
325. B. H. Lisko and S. N. Zilles. Specification techniques for data abstraction. *IEEE Trans. Software Eng.*, SE-1, March 1975.
326. A. Lötzbeyer. Simulation of a steam boiler. In J.-R. Abrial, E. Börger, and H. Langmaack (eds.), *Formal Methods for Industrial Applications. Specifying and Programming the Steam-Boiler Control*, Lecture Notes in Computer Science, Vol. 1165, pp. 493–499. Springer-Verlag, 1996.
327. M. Maia and R. Bigonha. An ASM-Based Approach for Mobile Systems. Technical Report LLP-12/99, Programming Language Laboratory, Computer Science Department, Universidade Federal de Minas Gerais, Brasil, 1999.  
Using the Interacting ASM techniques introduced in [328], the authors describe the use of ASMs to specify the semantics of active mobile objects. Mobility is expressed by dynamic changes in the communication topology. An earlier version appears as Technical Report LP 002/99 (same institution).
328. M. Maia, V. Iorio, and R. Bigonha. Interacting Abstract State Machines. In U. Glässer and P. Schmitt (eds.), *Proc. 5th Int. Workshop on Abstract State Machines*, pp. 37–49. Magdeburg University, 1998.  
An extended abstract describing an extension to ASMs supporting the interaction of independent ASM agents by means of message passing. The full version appears as M. Maia and R. Bigonha, *Formal Semantics for Interactive Abstract State Machine Language*, Technical Report RT 005/98, Universidade Federal de Minas Gerais, Brazil, 1998. Continued in [327].
329. K. Mani Chandy and J. Misra. *Parallel Program Design. A Foundation*. Addison Wesley, 1988.
330. W. May. Specifying complex and structured systems with evolving algebras. In *TAPSOFT’97: Theory and Practice of Software Development, 7th Int. Joint Conf. CAAP/FASE*, Lecture Notes in Computer Science, Vol. 1214, pp. 535–549. Springer-Verlag, 1997.  
An approach is presented for specifying structured systems with ASMs by means of aggregation and composition. An earlier version appeared under the title “Modeling Rule-Based and Structured Systems with Evolving Algebras” as Technical Report, Freiburg, 1996. For some of the structuring concepts defined here, simpler definitions are given in [134] which are geared

- to their natural integration into the basic parallelism of multiple simultaneous machine actions of ASMs.
331. P. J. McCann and G.-C. Roman. Programming abstractions for mobile computing. *IEEE Trans. Software Eng.*, 24(2):97–110, 1998.
  332. L. Mearelli. Refining an ASM specification of the production cell to C++ code. *J. Universal Computer Science*, 3(5):666–688, 1997.  
Source code for the ASM specification of the Production Cell described in [120]. For the generation of this code see [391].
  333. B. Meyer. *Eiffel: The Language*. Prentice-Hall, 1992.
  334. J. M. Morris. A theoretical basis for stepwise refinement. *Science of Computer Programming*, 9(3), 1987.
  335. M. Mohnen. A compiler correctness proof for the static link technique by means of evolving algebras. *Fundamenta Informatica*, 29(3):257–303, 1997.  
The static link technique is a common method used by stack-based implementations of imperative programming languages. The author uses ASMs to prove the correctness of this well-known technique in a non-trivial subset of Pascal.
  336. E. F. Moore. The shortest path through a maze. In *Proc. Int. Sympos. on Theory of Switching*, The Annals of the Computation Laboratory of Harvard University, Vol. 30.II. Harvard University Press, 1959.
  337. C. C. Morgan. *Programming from Specifications*. Prentice-Hall, 1990,<sup>2</sup>1994.
  338. J. Morris. *Algebraic Operational Semantics and Modula-2*. PhD thesis, University of Michigan, Ann Arbor, Michigan, 1988.  
Thesis supervised by Gurevich. The earliest ASM formalization of a programming language. The semantical description is parse-tree directed, but flat. An extended abstract appeared as Y. Gurevich and J. Morris, “Algebraic Operational Semantics and Modula-2”, in E. Börger, H. Kleine Büning and M. M. Richter, eds., *CSL’87, 1st Workshop on Computer Science Logic*, Lecture Notes in Computer Science, Vol. 329, pp. 81–101, Springer-Verlag, 1988.
  339. J. Morris and G. Pottinger. Ada-Ariel semantics. Odyssey Research Associates, unpublished manuscript, July 1990.
  340. Y. N. Moschovakis. What is an algorithm? In B. Engquist and W. Schmid (eds.), *Mathematics Unlimited – 2001 and beyond*, pp. 919–936. Springer-Verlag, 2001.
  341. L. S. Moss and D. E. Johnson. Dynamic interpretations of constraint-based grammar formalisms. *J. Logic, Language, and Information*, 4(1):61–79, 1995.  
Extends the work of [297] to grammar formalisms based on Kasper–Rounds logics. See [342].
  342. L. S. Moss and D. E. Johnson. Evolving algebras and mathematical models of language. In L. Polos and M. Masuch (eds.), *Applied Logic: How, What, and Why*, Synthese Library, Vol. 626, pp. 143–175. Kluwer Academic Publishers, 1995.  
Extends the work of [297] to several other grammar formalisms.
  343. P. D. Mosses. *Action Semantics*. Cambridge University Press, Cambridge, 1992.

344. W. Mueller, R. Dömer, and A. Gerstlauer. The formal execution semantics of SpecC. Technical Report TR ICS 01-59, Center for Embedded Computer Systems at the University of California at Irvine, 2001.  
Adapting the async ASM model of VHDL in [111, 112] and the work in [345], an async ASM model for the semantics of SpecC is developed which covers the execution of SpecC behaviors and their interaction with the simulation kernel. This includes wait, waitfor, par, pipe, and try statements.
345. W. Mueller, J. Ruf, D. W. Hoffmann, J. Gerlach, T. Kropf, and W. Rosenstiehl. The simulation semantics of SystemC. In *Proc. Design Automation and Test in Europe (DATE 2001)*, pp. 64–70, IEEE CS Press, March 2001.  
Adapting the distributed ASM model of VHDL in [111, 112], a distributed ASM model for the semantics of SystemC is developed which covers method, thread, clocked thread behavior, and their interaction with the simulation kernel. Watching statements, signal assignment and wait statements are formalized for version V1.0 of SystemC. An extended version will appear in [346].
346. W. Mueller, J. Ruf, and W. Rosenstiel. An ASM-based semantics of systemC simulation. In W. Mueller, J. Ruf, and W. Rosenstiel (eds.), *SystemC - Methodologies and Applications*. Kluwer Academic Publishers, 2003.  
See [345].
347. B. Müller. *Eine objektorientierte Prolog-Erweiterung zur Entwicklung wissensbasierter Systeme*. PhD thesis, University of Oldenburg, Germany, 1994.  
Thesis supervised by Appelrath and Börger. Defines an object-oriented extension of Prolog to be applied for the development of knowledge based systems. The semantics is defined (in Chap. 5) as an extension of Börger’s Prolog model [75].
348. B. Müller. A semantics for hybrid object-oriented Prolog systems. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 428–433, Elsevier, Amsterdam, 1994.  
On Börger’s suggestion this work extends the rules given in [75] for the user-defined core of Prolog to define the semantics of a hybrid object-oriented Prolog system. The definition covers the central object-oriented features of object creation and deletion, data encapsulation, inheritance, messages, polymorphism and dynamic binding. See [347].
349. W. Müller. *Executable Graphics for VHDL-Based Systems Design*. PhD thesis, University of Paderborn, Germany, 1996.  
Uses ASMs to define the behavioral semantics of PHDL, a pictorial extension of VHDL’93. The ASMs for VHDL defined in [111, 112] are reused.
350. M. Müller-Olm. *Modular Compiler Verification. A Refinement-Algebraic Approach Advocating Stepwise Abstraction*, Lecture Notes in Computer Science, Vol. 1283. Springer-Verlag, 1997.  
The author’s PhD thesis. The considered language is a sublanguage of Occam with real-time features. See also the PROCOS II Esprit Basic Research 7071 Report MMO 12/3 (1996), University of Kiel: Structuring Code Generator Correctness Proofs by Stepwise Abstracting the Machine Language’s Semantics.
351. Z. Németh. Definition of a parallel execution model with Abstract State Machines. *Acta Cybernetica*, 15(3):417–455, 2002.  
Two ASMs are defined and related by a refinement correctness proof, as preparation for designing and verifying a distributed parallel Prolog execution model.

352. Z. Németh and V. Sunderam. A formal framework for defining grid systems. In *Proc. Int. Sympos. on Cluster Computing and the Grid (CCGrid2002)*, pp. 202–211, Berlin, 21–24 May 2002. IEEE Computer Society Press.  
ASMs are used to define a model for grid systems.
353. M. Nicolosi-Asmundo and E. Riccobene. Consistent integration for sequential Abstract State Machines. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 324–340. Springer-Verlag, 2003.  
Two operations to compose basic ASMs are defined and illustrated by two case studies.
354. A. Nowack. Deciding the verification problem for Abstract State Machines. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 341–355. Springer-Verlag, 2003.
355. I. Ober. More meaningful UML models. In *Proc. TOOLS*, pp. 146–157, Sydney, Australia, 20–23 November 2000. IEEE Computer Society Press.  
ASMs are used to define an executable semantics for UML which covers real-time aspects. The work is inspired by the ASM model for SDL in [222] and uses the ASM Workbench [170].
356. I. Ober. An ASM semantics for UML derived from the meta-model and incorporating actions. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 356–371. Springer-Verlag, 2003.
357. M. Odersky. Programming with variable functions. In *ICFP’98, Proc. 3rd ACM SIGPLAN Int. Conf. on Functional Programming*, ACM SIGPLAN Notices, Vol. 34 (1), pp. 105–116, January 1999.  
The use of “variable functions” (functions which can be updated at specified points in their domains) is proposed as a method for deriving efficient imperative programs from functional programs. The notion of a variable function is drawn from the dynamic functions of ASMs.
358. C. Pahl. Towards an action refinement calculus for Abstract State Machines. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines – ASM 2000, Int. Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, TIK-Report, No. 87, pp. 326–340. Swiss Federal Institute of Technology (ETH) Zurich, March 2000.  
A refinement calculus for (a reformulation of) ASMs is presented.
359. C. Pair. Types Abstraites et Sémantique Algébrique des Langages de Programmation. Technical Report TR 80-R-011, Centre de Recherche en Informatique de Nancy, France, 1980.  
Often referred to as the first publication where the idea is formulated that the most general notion of state of computing systems is Tarski’s notion of structures. See [210]. The similarity between data types and algebras is however already observed in [325, 265].
360. D. L. Parnas and J. Madey. Functional documents for computer systems. *Sci. of Comput. Program.*, 25:41–62, 1995.
361. B. Pehrson and I. Simon. I: Technology/foundations. In *IFIP 13th World Computer Congress 94*, Elsevier, Amsterdam, 1994.

- Stream C (Evolving Algebras) (pp. 377–441), organized by Gurevich, contains short versions of the talks presented to the first international ASM workshop; see [39, 56, 76, 97, 107, 118, 228, 246, 348, 365, 376].
362. C. N. Plonka. *Model Checking for the Design with Abstract State Machines*. Diplom thesis, CS Department of University of Ulm, Germany, January 2000.  
A feasibility study, carried out upon Börger’s suggestion at Siemens Research, of model checking ASMs for two industrial case studies: the Production Cell [120] and a statistical multiplexing unit. An error was detected in [120] concerning a refinement step for the deposit belt, due to an erroneous (easily repaired) symmetry assumption made during the specification for the unloading actions of feedbelt, press and deposit belt. Due to additional scheduling assumptions, made for the model checking of the Production Cell ASM in [424] to guarantee maximal performance of the model, the mistake had remained undiscovered there.
363. A. Poetzsch-Heffter. Interprocedural data flow analysis based on temporal specifications. Technical Report 93-1397, Cornell University, Ithaca, New York, 1993.  
Investigates the specification of data flow problems by temporal logic formulas and proves fixpoint analyses correct. Temporal formulas are interpreted with respect to programming language semantics given in the framework of ASMs.
364. A. Poetzsch-Heffter. Comparing action semantics and evolving algebra based specifications with respect to applications. In *Proc. 1st Int. Workshop on Action Semantics*, pp. 43–47, 1994.  
Action semantics is compared to ASM based language specifications. In particular, different aspects relevant to language documentation and programming tool development are discussed.
365. A. Poetzsch-Heffter. Deriving partial correctness logics from evolving algebras. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 434–439, Elsevier, Amsterdam, 1994.  
A proposal for deriving partial correctness logics from simple ASM models of programming languages. A basic axiom (schema) is derived from an ASM and is used to obtain more convenient logics. See [405].
366. A. Poetzsch-Heffter. Prototyping realistic programming languages based on formal specifications. *Acta Informatica*, 34:737–772, 1997.  
A tool supporting the generation of language-specific software from specifications is presented, enabling in particular the generation and refinement of interpreters based on formal language specifications. Static semantics is defined by an attribution technique (e.g. for the specification of flow graphs). The dynamic semantics is defined by ASMs. As an example, an object-oriented programming language with parallelism is specified. Part of this work has appeared as Report TR 93-1396, Cornell University and in 1994 as *Developing Efficient Interpreters based on Formal Language Specifications* in P. Fritzson (ed.): *Compiler Construction, Lecture Notes in Computer Science*, Vol. 786, pp. 233–247, Springer-Verlag, 1994.
367. A. Prinz. *Formal Semantics for SDL. Definition and Implementation*. Habilitationsschrift, Humboldt University of Berlin, Germany, 2000.  
Contains a complete definition and implementation of the static and dynamic semantics of a characteristic sublanguage of SDL.
368. A. Prinz and B. Thalheim. Operational semantics of transactions. In X. Zhou and K.-D. Schewe (eds.), *Proc. 14th Australian Database Conf. (ADC2003)*,

Australian Computer Science Commun., Vol. 25(2), pp. 169–179. Australian Computer Society, 2003.

Defines an ASM model for database transactions which is instantiated for the in-private and the in-place setting and used to explain the constraint enforcement used in SQL'99.

369. C. Pusch. Verification of compiler correctness for the WAM. In J. von Wright, J. Grundy, and J. Harrison (eds.), *Theorem Proving in Higher Order Logics (TPHOLs'96)*, Lecture Notes in Computer Science, Vol. 1125, pp. 347–362. Springer-Verlag, 1996.

See comment to [132].

370. H. Reichel. Unifying ADT and evolving algebra specifications. *Bull. EATCS*, 59:112–126, 1996.

Di-algebras, a notion unifying algebras and co-algebras, are used to combine algebraic specifications of abstract data types with ASMs. A characterization of ASMs as terminally constraint Di-algebras is introduced to justify the co-induction proof principle for ASMs. Also a Di-algebra thesis is stated as the algebraic counterpart of the ASM thesis.

371. W. Reisig. Petri nets in software engineering. In W. Brauer, W. Reisig, and G. Rozenberg (eds.), *Petri Nets: Applications and Relationships to other Models of Concurrency*, Lecture Notes in Computer Science, Vol. 255, pp. 63–96. Springer-Verlag, 1987.

372. W. Reisig. *Elements of Distributed Algorithms*. Springer-Verlag, 1998.

373. G. R. Renardel de Lavalette. A logic of modification and creation. In C. Condonavdi and G. R. Renardel de Lavalette (eds.), *Logical Perspectives on Language and Information*. CSLI publications, Stanford, CA, 2001.

374. E. Riccobene. A formal computational model for PANDORA. Technical Report CSTR-92-16 and ACRC-92-15, University of Bristol, Department of Computer Science, 1992.

The ASM model for Parlog developed in [123] is extended by the don't-know non-determinism of Pandora.

375. E. Riccobene. *Modelli Matematici per Linguaggi Logici*. PhD thesis, University of Catania, Sicily, Italy, Academic year 1991/92.

Systematic treatment of ASM models for Gödel [124], Parlog [123], Pandora [374], Concurrent Prolog [122], GHC. Thesis supervised by Börger.

376. D. Rosenzweig. Distributed computations: Evolving algebra approach. In B. Pehrson and I. Simon (eds.), *IFIP 13th World Computer Congress*, Vol. I: Technology/Foundations, pp. 440–441, Elsevier, Amsterdam, 1994.

Remarks on some ASM models of concurrent and parallel computation.

377. D. Rosenzweig, D. Runje, and N. Slani. Privacy, abstract encryption and protocols: an ASM model—part I. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003—Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 372–390. Springer-Verlag, 2003.

Provides an AsmL executable model of abstract encryption.

378. H. Rust. Hybrid Abstract State Machines: Using the hyperreals for describing continuous changes in a discrete notation. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines – ASM 2000, International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, TIK-Report, No. 87, pp. 341–356. ETH Zürich, March 2000.

- A hybrid version of ASMs, incorporating the hyperreals for continuously changing quantities, is described.
379. H. Rust. *A non-standard approach to operational semantics for timed systems*. Habilitation thesis, BTU Cottbus, Germany, 2002.
- Time moments are defined as multiples of some arbitrary, but fixed infinitesimal, allowing us to model real-time system behavior with infinitesimal exactness and refinements of actions. ASMs are used to model hybrid systems, adding interleaving to the synchronous parallel execution model.
380. H. Sasaki. A formal semantics for Verilog-VHDL simulation interoperability by Abstract State Machines. In *Proc. IEEE Conf. DATE'99 (Design, Automation and Test in Europe), ICM Munich, Germany*, pp. 353–357, 9–12 March 1999.
- Based upon the VHDL models developed in [111, 112], a formal semantics for Verilog-HDL and VHDL focusing on the simulation model (with signal scheduling and time control) is defined. The semantics presented is faithful to the language reference manual and is proposed as a first step towards semantic interoperability analysis on multi-semantic domains such as Verilog-AMS and VHDL-AMS. Extended in [381].
381. H. Sasaki. A formal semantics on net delay in Verilog-HDL. In *Proc. Asia Pacific Conf. on Chip Design Languages (APCHDL'99)*, pp. 100–106, Fukuoka, Japan, 6–8 October 1999.
- An extension of [380] giving semantics for net delays in Verilog-HDL using ASMs.
382. H. Sasaki. A new dynamic equation scheduling to extend VHDL-AMS. In *Proc. Asia Pacific Conf. on Chip Design Languages (APCHDL'99)*, pp. 47–52, Fukuoka, Japan, 6–8 October 1999.
- An extension to VHDL-AMS for dynamic equation scheduling is proposed. The semantics of the extension is given in terms of the ASM model for VHDL-AMS presented in [383].
383. H. Sasaki, K. Mizushima, and T. Sasaki. Semantic validation of VHDL-AMS by an Abstract State Machine. In *Proc. BMAS'97 (IEEE/VIUF Int. Workshop on Behavioral Modeling and Simulation)*, pp. 61–68, Arlington, VA, 20–21 October 1997.
- An extension of the ASM model defined for VHDL in [111, 112] to provide a rigorous definition of VHDL-AMS, following the IEEE Language Reference Manual for the analog extension of VHDL. For an extension see [384]. See also [381, 382, 380].
384. T. Sasaki, H. Sasaki, and K. Mizushima. Semantic analysis of VHDL-ASM by attribute grammar. In *Proc. FDL'98 (Forum on Design Languages), Lausanne, Switzerland*, pp. 123–131, 6–10 September 1998.
- An extension of [383] to provide a formal semantics of the VHDL Analog Mixed Signal extension by means of attribute grammars. The formulation treats both the static and the dynamic aspects of semantics and permits one to show the equality of process behavior.
385. J. Sauer. *Wissensbasiertes Lösen von Ablaufplanungsproblemen durch explizite Heuristiken*. PhD thesis, Universität Oldenburg, Germany, 1993.
- Published in: Dissertationen zur Künstlichen Intelligenz, Vol. 37, Infix-Verlag, Dr. Ekkehardt Hundt, St. Augustin, 1993. Uses ASMs to define the semantics for the HERA language (and its implementation in Prolog), a special-purpose

programming language for the representation and manipulation of scheduling knowledge on the basis of heuristics, tailored to program efficient and reusable scheduling algorithms for production planning and control. See also J. Sauer, “Evolving Algebras for the Description of a Meta-Scheduling System”, in H. Kleine Büning, ed., *Workshop der GI-Fachgruppe Logik in der Informatik*, Technical Report TR-RI-94-146, Universität Paderborn, 1994.

386. G. Schellhorn. *Verifikation abstrakter Zustandsmaschinen*. PhD thesis, Universität Ulm, Germany, 1999.
- ASMs are embedded into dynamic logic. Two refinement notions are extracted from typical ASM refinements and formalized in dynamic logic. A general modularisation theorem is proved for schemes to prove the correctness of refinements. An improved version of this theorem appears in [387]. The KIV system is enhanced to apply those proof techniques for a KIV verification of the WAM correctness proof in [132]. An English version of the thesis is available at Schellhorn’s web site.
387. G. Schellhorn. Verification of ASM refinements using generalized forward simulation. *J. Universal Computer Science*, 7(11):952–979, 2001.
- See [386].
388. G. Schellhorn and W. Ahrendt. Reasoning about Abstract State Machines: The WAM case study. *J. Universal Computer Science*, 3(4):377–413, 1997.
- The Karlsruhe Interactive Verifier (KIV system) is applied to mechanically verify the proof of correctness of the Prolog to WAM transformation described in [132]. The starting point was the Diplom Thesis *Von Prolog zur WAM. Verifikation der Prozedurübersetzung mit KIV* of W. Ahrendt, Universität Karlsruhe (Germany) 1995. See comment to [132] and [389, 386].
389. G. Schellhorn and W. Ahrendt. The WAM case study: Verifying compiler correctness for Prolog with KIV. In W. Bibel and P. Schmitt (eds.), *Automated Deduction – A Basis for Applications*, Vol. III: Applications, pp. 165–194. Kluwer Academic Publishers, 1998.
- Continuation of [388].
390. J. Schmid. Executing ASM specifications with AsmGofer. Web pages at <http://www.tydo.de/AsmGofer>.
- The web site for the machine to execute, equipped with graphical user interface, ASMs which are enhanced with the structuring and composition concepts defined in [134]. AsmGofer executes the Light Control ASM <http://www.tydo.de/AsmGofer/light> defined in [125] and all the ASMs defined in [406]. In [152] AsmGofer has been used to build a simulator for UML state diagrams.
391. J. Schmid. Compiling Abstract State Machines to C++. *J. Universal Computer Science*, 7(11):1069–1088, 2001.
- Introduces a scheme for compiling ASMs from the syntax of the ASM Workbench [170] to C++, coding algebraic types, pattern matching, functional expressions, dynamic functions and simultaneous updates in such a way that efficient C++ code is obtained without losing the structure of the original ASM specification. The compiler has been successfully applied in the FALKO project at Siemens [121]. In an early application C++ code was generated from a translation of the Production Cell ASM in [120] to the ASM Workbench format ASM-SL [170]. An HTML version is available at <http://www.tydo.de/ProductionCell/>.

392. J. Schmid. *Refinement and Implementation Techniques for Abstract State Machines*. PhD thesis, University of Ulm, Germany, 2002.  
Thesis supervised by Börger and located at Siemens Corporate Research in München from August 1998 to July 2000. The thesis enriches ASMs by structuring and composition concepts [134] and their implementation in the AsmGofer system, developed for executing ASMs in an environment with a graphical user interface. The concepts have been successfully applied in a middle-sized software development project at Siemens [121, 391], in the Light Control Case Study [125], in an industrial ASIC design and verification project (including a compiler from ASM to VHDL), and for the modeling and implementation of Java and the JVM in [406]. Electronic version available at <http://www.tydo.org/files/papers/dissJS.pdf>.
393. P. Schmitt. Proving WAM compiler correctness. Technical Report 33/94, Universität Karlsruhe, Fakultät für Informatik, Germany, 1994.  
Feasibility analysis of Börger’s proposal to the DFG project “Deduktion” to mechanize the Prolog-to-WAM compiler correctness proof in [132]. See [388, 387, 386, 369].
394. A. Schönegge. Extending dynamic logic for reasoning about evolving algebras. Technical Report 49/95, Universität Karlsruhe, Fakultät für Informatik, Germany, 1995.  
EDL, an extension of dynamic logic, is presented, which permits one to directly represent statements about ASMs. Such a logic lays the foundation for extending KIV (Karlsruhe Interactive Verifier) to reason about ASMs directly. See [405].
395. W. Schönfeld. Interacting Abstract State Machines. In U. Glässer and P. Schmitt (eds.), *Proc. 5th Int. Workshop on Abstract State Machines*, pp. 22–36. Magdeburg University, 1998.  
An extension to ASMs which permits one to specify forced synchronization of agent moves (à la Petri nets) is proposed and explored on some examples.
396. A. Schönhage. Storage modification machines. *SIAM J. Comp.*, 9:490–508, 1980.  
Shown in [177] to be equivalent to a class of unary sequential ASMs.
397. M. Schrefl and G. Kappel. Cooperation contracts. In T. J. Teorey (ed.), *Proc. 10th Int. Conf. on the Entity Relationship Approach (ER’91)*, pp. 285–307, San Mateo, California, 23–25 October 1991. Entity Relationship Institute.  
The authors introduce the concept of *cooperative* message handling where multiple objects can establish cooperation contracts governing their answers to jointly received messages. An ASM rule is defined (Fig. 9, p. 304) to formalize the run-time search of the most specific cooperation contract which implements a cooperative message. See [232].
398. D. Scott. Definitional suggestions for automata theory. *J. Computer and System Sciences*, 1:187–212, 1967.
399. M. Shaw and D. Garlan. Formulations and formalisms in software architecture. In J. van Leeuwen (ed.), *Computer Science Today: Recent Trends and Developments*, Lecture Notes in Computer Science, Vol. 1000. Springer-Verlag, 1995.
400. I. Soloviev. *Exploration and Experimental Implementation of Recursive Patterns and Functions Embedding Into Prolog Language Syntactical Environment*. PhD thesis, St. Petersburg University, Russia, 1995.

In Russian. A functional extension of Prolog with a specialized unification algorithm is proposed. ASMs are used to define the operational semantics of the language.

401. M. Spielmann. Automatic verification of Abstract State Machines. In N. Halbwachs and D. A. Peled (eds.), *Proc. 11th Int. Conf. on Computer-Aided Verification (CAV '99)*, Lecture Notes in Computer Science, Vol. 1633, pp. 431–442. Springer-Verlag, 1999.

A class of restricted ASM programs is introduced, along with a PSPACE algorithm for verifying the correctness of certain CTL\*-like temporal-logic properties of such programs. The limits on the verifiability of generalizations of this class are discussed.

402. M. Spielmann. *Abstract State Machines: Verification Problems and Complexity*. PhD thesis, University of Aachen, Germany, 2000.

Investigation of the complexity of decision problems for certain classes of ASMs. Most of the results appear in [401, 404, 403]. The second part of the thesis relates to the work in [62]. A restricted ASM model to capture log-space computable functions on structures is defined; see also [235].

403. M. Spielmann. Model checking Abstract State Machines and beyond. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 323–340. Springer-Verlag, 2000.

Decision problems for ASMs are investigated, i.e. problems to decide, for an ASM  $M$  of a given class and for a property  $P$  of a given form, whether  $M$  satisfies  $P$ . For particular classes of machines and of property describing formulae, the computational complexity of such problems is studied for the following two cases: (a) given  $M$ ,  $P$  and input  $I$ , decide whether  $P$  holds during all  $M$ -computations over  $I$  (called model-checking problem); (b) given  $M$  and  $P$ , decide whether for every input  $I$ ,  $P$  holds during all  $M$ -computations over  $I$  (called the verification problem). Appeared also as TIK-Report No. 87, pp. 357–375, ETH Zürich, March 2000.

404. M. Spielmann. Verification of relational transducers for electronic commerce. In *Proc. 19th ACM Sympos. Principles of Database Systems (PODS 2000)*, pp. 92–103, Dallas, Texas, 2000. ACM Press.

An investigation into decision problems for certain transaction protocols specifying the interaction of multiple parties, each equipped with an active database. Inspired by the relational transducers in [1], ASM-transducers are defined and shown to have various solvable decision problems.

405. R. F. Stärk and S. Nanchen. A logic for Abstract State Machines. *J. Universal Computer Science*, 7(11):981–1006, 2001.

A new logic for sequential, non-distributed ASMs is presented which is based on an atomic predicate for function updates and on a definedness predicate for the termination of the evaluation of ASM rules. The logic allows for sequential and hierarchical recursive submachine composition as defined in [134]. It is proven complete for hierarchical non-recursive ASMs. This logic provides a unifying view of the logics for ASMs developed in [238, 394, 365, 207]. A preliminary version appeared in L. Fribourg (ed.): *Computer Science Logic (CSL 2001)*, Lecture Notes in Computer Science, Vol. 2142, pp. 217–231, Springer-Verlag, 2001.

406. R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation*. Springer-Verlag, 2001.

A high-level description, together with a mathematical and an experimental analysis (verification and validation), of Java and of the Java Virtual Machine (JVM), including a standard compiler of Java programs to JVM code and the security-critical bytecode verifier component of the JVM. Includes an executable ASM specification written for AsmGofer. For an evaluation see [272, Sect. 6.2]; for more information see <http://www.inf.ethz.ch/~jbook/>.

407. M. M. Stegmüller. *Formale Verifikation des DLX RISC-Prozessors: Eine Fallstudie basierend auf abstrakten Zustandsmaschinen*. Diplom thesis, University of Ulm, Germany, 1998.

PVS is used for the formal verification of the parallelization of the ASM specification for the serial DLX architecture, the first step of the refinement to its parallel version with five-stage pipelining in [119].

408. K. Stenzel. Verification of JavaCard programs. Technical report 2001-5, Institut für Informatik, Universität Augsburg, Germany, 2001.

Available at <http://www.Informatik.Uni-Augsburg.DE/swt/fmg/papers/>. The report is about the formal verification of JavaCard or sequential Java programs (i.e. without synchronized statements). A calculus in dynamic logic is defined and implemented in KIV. KIV parses the original JavaCard or Java program, resolves names and types in the same manner as a normal Java compiler, and produces an annotated abstract syntax tree that is the input for the verification. All sequential Java statements are supported, including exceptions, breaks, static initialization, objects, dynamic method lookup and arrays. The abstract syntax of Java programs, the proof rules, and the underlying algebraic specifications for the object store and the primitive data types, and a formal semantic is described in detail. An example proof and a list of validation programs conclude the report. For information on preliminary work on formalizing ASM models for Java in KIV see <http://www.informatik.uni-augsburg.de/swt/fmg/applications/>.

409. K. Stroetmann. The constrained shortest path problem: A case study in using ASMs. *J. Universal Computer Science*, 3(4):304–319, 1997.

Upon Börger’s suggestion, an abstract, non-deterministic form of the constrained shortest path problem is defined as an ASM and proven correct, and then refined to the level of implementation.

410. A. Sünbül. *Architectural Design of Evolutionary Software Systems in Continuous Software Engineering*. PhD thesis, TU Berlin, Germany, 2001.

Taking up a suggestion in [86, Sect. 4] this dissertation develops a language for specifying software systems by linking components via connectors. Components are abstractly characterized by the services they import and export which are defined by high-level specifications (possibly depending on given views) and have to satisfy certain constraints on well-formedness and on the ordering of usage (called use structure). For connectors, which connect services required in one component to services offered by other components, a refinement concept is defined. ASM rules are provided to check the consistency of software architectures developed in that language, namely checking componentwise (a) for each imported service its correct connection to a corresponding exported service (with respect to signature and specification), (b) for each exported service that the imported services it uses satisfy the constraints of the used components, and (c) that the (optional) refinement is correct with respect to the system constituents (types, views, components, connectors). The proposed machine has been made executable in XASM. Extended abstracts of some of the ideas in the thesis have been published by M.

Anlauff and A. Sünbül as *Software Architecture Based Composition of Components* (Gesellschaft für Informatik, Sicherheit und Zuverlässigkeit software-basierter Systeme, May 1999), *Component Based Software Engineering for Telecommunication Software* (Proc. SCI/ISAS Conf., Orlando, Florida 1999), *Domain-Specific Languages in Software Architecture* (Proc. Integrated Design and Process Technology IDPT99, June 1999).

411. J. Teich. Project Buildabong at University of Paderborn. <http://www-date.upb.de/RESEARCH/BUILDABONG/buildabong.html>, 2001.
- The project, led by Teich at the University of Paderborn, uses ASMs to provide behavioral and structural descriptions of application-specific instruction set processors, from which (using XASM [15] and Gem-Mex [18]) bit-true and cycle-accurate simulators and debuggers are derived. See the paper “Design Space Characterization for Architecture/Compiler Co-Exploration” by D. Fischer, J. Teich, R. Weper, U. Kastens, M. Thies in: Proc. ACM Conf. CASES’01, 16–17 November, 2001, Atlanta, Georgia, USA.
412. J. Teich, P. Kutter, and R. Weper. Description and simulation of microprocessor instruction sets using ASMs. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 266–286. Springer-Verlag, 2000.
- A method for transforming register transfer descriptions of microprocessors into executable ASM specifications is described and illustrated using the ASM model developed in [286] for the ARM2 RISC processor. The description exploits the natural correspondence between the simultaneous execution of all guarded update rules of an ASM and a single-clock hardware step executing a set of Leuper’s guarded register transfer patterns. XASM [15] is used together with the Gem-Mex tool [18] which generates a graphical simulator for the given architecture. See also [413]. Also appears in TIK-Report No. 87, pp. 376–397, ETH Zürich, March 2000.
413. J. Teich, R. Weper, D. Fischer, and S. Trinkert. A joint architecture/compiler design environment for ASIPs. In *Proc. Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES2000)*, pp. 26–33, San Jose, CA, USA, November 2000. ACM Press.
- An ASM model is developed for a VLIW digital signal processor of the Texas Instruments TMS320 C6200 family to illustrate the Buildabong method [411]. See also [412].
414. H. Tonino. *A Theory of Many-sorted Evolving Algebras*. PhD thesis, Delft University of Technology, Netherlands, 1997.
- Based on a two-valued many-sorted logic of partial functions (with a complete and sound Fitch-style axiomatization), a structural operational and a Hoare-style axiomatic semantics is given for many-sorted non-distributed deterministic ASMs. The SOS semantics is defined in two levels, one for the sequential and one for the parallel ASM constructs. Two (sound but not complete) Hoare-style descriptions are given, one for partial and one for total correctness. A first part appeared under the title “A Formalization of Many-sorted Evolving Algebras” as Report TR 93-115 at TU Delft. An extended abstract appeared under the title *A Sound and Complete SOS-Semantics for Non-Distributed Deterministic Abstract State Machines* in [226, pp. 91–110].
415. H. Tonino and J. Visser. Stepwise refinement of an Abstract State Machine for WHNF-reduction of  $\lambda$ -terms. Technical Report 96–154, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Netherlands, 1996.

- A series of ASMs for finding the weak head normal form (WHNF) of an arbitrary term of the  $\lambda$ -calculus is presented.
416. K. J. Turner (ed.). *Using Formal Description Techniques. An Introduction to Estelle, LOTOS and SDL*. John Wiley, New York, 1993.
417. M. Vale. The evolving algebra semantics of COBOL. Part I: Programs and control. Technical Report CSE-TR-162-93, EECS Dept., University of Michigan, 1993.
- An ASM for the control constructs of COBOL. A description of a plan for a series of ASMs for all of COBOL is sketched (but not carried out). Missing constructs concern source text manipulations, report writer, communication, debug, and segmentation modules.
418. J. Visser. *Evolving Algebras*. Master's thesis, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, Netherlands, 1996.
- The monad programming method is used to write a compiler/run-analyzer for ASMs in Gofer. Static functions can be supplied to the ASMs by means of Gofer functions.
419. F. von Henke. Putting software technology to work. In K. Duncan and K. Krueger (eds.), *IFIP 13th World Computer Congress 1994*, pp. 345–350. Elsevier, 1994.
420. C. Wallace. The semantics of the C++ programming language. In E. Börger (ed.), *Specification and Validation Methods*, pp. 131–164. Oxford University Press, 1995.
- The description in [251] of the semantics of C is extended to C++.
421. C. Wallace. The semantics of the Java programming language: Preliminary version. Technical Report CSE-TR-355-97, EECS Dept., University of Michigan, December 1997.
- A specification of the static and dynamic semantics of Java, using ASMs and Montages. This work showed the shortcomings of the original formulation of Montages [311] and led to its state machine based reformulation in [19]. See [309] and the independent earlier Java modeling work [138] which was continued in [137, 139, 140, 141] and [406]. See also [23].
422. C. Wallace, G. Tremblay, and J. N. Amaral. An Abstract State Machine specification and verification of the location consistency memory model and cache protocol. *J. Universal Computer Science*, 7(11):1089–1113, 2001.
423. P. Wegner. Why interaction is more powerful than algorithms. *Commun. ACM*, 40:80–91, 1997.
424. K. Winter. Model checking for Abstract State Machines. *J. Universal Computer Science*, 3(5):689–701, 1997.
- Inspired by Börger's lectures on ASMs in Freiburg in the Fall of 1994, Winter develops a framework for using a model checker to verify ASM specifications. It is applied to the production cell control model described in [120]. See [362] for an interesting problem with refining abstractions for model checking purposes. For an extension see [175, 425, 426, 427].
425. K. Winter. Towards a methodology for model checking ASM: Lessons learned from the flash case study. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines: Theory and Applications*, Lecture Notes in Computer Science, Vol. 1912, pp. 341–360. Springer-Verlag, 2000.

A general discussion of applying model checking to ASMs. Following a suggestion by Börger, the ASM specification of the FLASH cache coherence protocol [189] is checked using SMV as a case study. An extension of [175, 424]. Also appears in TIK-Report No. 87, pp. 398–425, ETH Zürich, March 2000.

426. K. Winter. *Model Checking Abstract State Machines*. PhD thesis, Technical University of Berlin, Germany, 2001.

Based upon [424, 175, 425, 427], a transformation of ASMs to FSMs and abstraction mechanisms in the context of model checking large ASMs are investigated and implemented. The underlying tools are the ASM Workbench [170], SMV and Multiway Decision Graphs (for the latter see also [427]).

427. K. Winter. Model checking with abstract types. In S. D. Stoller and W. Visser (eds.), *Workshop on Software Model Checking*, Electronic Notes in Theoretical Computer Science, Vol. 55 (3), Paris, France, July 23 2001. Elsevier Science B.V.

Investigates an interface from ASMs to Multiway Decision Graphs. See also the Report TR 01-16, Software Verification Research Center, The University of Queensland, November 2001.

428. K. Winter. Automated checking of control tables. E-mail to E. Börger, December 24, 2001.

Case study for automated checking of Control Tables, used by the Software Verification Research Centre at the University of Queensland, Australia, to specify railway interlocking systems. The control tables are formalized as ASMs and then transformed by the algorithm described in [175] to become input for the SMV model checker.

429. N. Wirth. Program development by stepwise refinement. *Commun. ACM*, 14(4), 1971.

430. J. Woodcock and M. Loomes. *Software Engineering Mathematics*. Pitman, 1988.

431. J. C. P. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, 1996.

432. A. Zamulin. Algebraic specification of dynamic objects. In *Proc. LMO'97 (Acte du Colloque Langage et Modeles a Objets)*, pp. 111–127, Paris, 22–24 October 1997. Edition Hermes.

A model for describing the behavior of dynamic objects is presented, using a state-transition system with the same semantics as (though not explicitly identified as) ASMs.

433. A. Zamulin. Specification of an Oberon compiler by means of a typed Gurevich machine. Technical Report 589.3945009.00007-01, Institute of Informatics Systems of the Siberian Division of the Russian Academy of Sciences, Novosibirsk, Russia, 1997.

A Typed Gurevich Machine [434] is used to define a compiler for Oberon to an algebraically-specified abstract target machine.

434. A. Zamulin. Typed Gurevich machines revisited. *Joint CS & IIS Bulletin, Computer Science*, 7:95–122, 1997.

An approach to combining type-structured algebraic specifications and ASMs is proposed. A preliminary version appeared in 1996 as preprint 36 of the Institute of Informatics Systems, Novosibirsk.

435. A. Zamulin. Object-oriented Abstract State Machines. In U. Glässer and P. Schmitt (eds.), *Proc. 5th Int. Workshop on Abstract State Machines*, pp. 1–21. Otto-von-Guericke-Universität Magdeburg, 1998.  
Proposes an extension of ASMs to include objects.
436. A. Zamulin. Specification of dynamic systems by typed Gurevich machines. In Z. Bubnicki and A. Grzech (eds.), *Proc. 13th Int. Conf. on System Science*, pp. 160–167, Wrocław, Poland, 15–18 September 1998.  
A combination of many-sorted algebraic specifications for states and ASM-rules for transitions is proposed as an approach for dynamic system specification. The approach is used in [433] to specify an Oberon compiler.
437. A. Zamulin. Generic facilities in object-oriented ASMs. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines – ASM 2000, Int. Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, TIK-Report, No. 87, pp. 426–446. ETH Zürich, March 2000.  
The object-oriented ASM framework introduced in [435] is extended to allow the definition of generic object types, type categories, functions, and procedures. Examples from the C++ Standard Template Library (STL) are provided. Previously appeared in Preprint 60, Institute of Informatics Systems, Siberian Division of the Russian Academy of Sciences, Novosibirsk, 1999.
438. A. Zamulin. Specifications in-the-large by typed ASMs. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele (eds.), *Abstract State Machines – ASM 2000, Int. Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, TIK-Report, No. 87, pp. 447–461. ETH Zürich, March 2000.  
A discussion of combining typed ASMs (as proposed in [436]) to produce larger ASMs.
439. W. Zimmerman and T. Gaul. On the construction of correct compiler back-ends: An ASM approach. *J. Universal Computer Science*, 3(5):504–567, 1997.  
The authors use ASMs to construct provably correct compiler back-ends based on realistic intermediate languages (and check the correctness of their proofs using PVS).
440. W. Zimmermann and A. Dold. A framework for modeling the semantics of expression evaluation with Abstract State Machines. In E. Börger, A. Gargantini, and E. Riccobene (eds.), *Abstract State Machines 2003–Advances in Theory and Applications*, Lecture Notes in Computer Science, Vol. 2589, pp. 391–406. Springer-Verlag, 2003.  
An ASM framework is defined for a uniform characterization of expression evaluation tactics used in programming languages like Fortran, ADA95, C, C++, Java, and C#.