2-1998

Abstract State Machines 1988-1998: Commented ASM Bibliography

James Huggins

Egon Borger

Abstract State Machines 1988-1998: Commented ASM Bibliography

Egon Börger and **James K. Huggins**

September 13, 2018

Abstract

This is the current version of an annotated bibliography of papers which deal with or use ASMs. It is compiled to a great extent from references and annotations provided by the authors of the listed papers and extends the annotated bibliography which previously appeared in [20]. Comments, additions and corrections are welcome and should be sent to `boerger@di.unipi.it` and `huggins@acm.org`[1].

Hartmut Ehrig asked the first author to write for this column what are the distinguishing features of the ASM approach to specification and verification of complex computing systems. In [21] an attempt had already been made to answer that question by discussing, in general comparative terms, some specific features which are characteristic for the ASM approach with respect to other well known approaches in the literature. That explanation seems to have been understood, as shown by the many positive reactions, but even more the numerous critical reactions of colleagues in the field who felt—rightly—that ASMs put justified doubt on cherished denotational, declarative, logical, functional and similar widespread beliefs in *pure*, i.e. not operational methods. Nevertheless some dissatisfaction remained with that paper because the discussion, in a sense unavoidably, remained in general terms which have been used during the last two or three decades again and again for the justification of many other methods.

The attempt to answer the question in a more *concrete* way led the two authors of this commented bibliography to systematically review again, revising and updating [20], what are the achievements and failures of ASM research since the discovery of the notion by Yuri Gurevich in 1988. What follows here is a way of answering Hartmut Ehrig's question; namely, we try to let the research results speak for the method.

If somebody really wants to know whether there is anything useful in the notion of ASM which has not been covered by competing methods in the literature, he or she should try out the method on a challenging (not a toy) specification or verification problem. We have no doubt that then it will become clear why so much successful research could be done in such a short period by a relatively small number of researchers, as documented in the commented bibliography below.

Current updates of this bibliography (as well as some of the papers listed below) will be available on the ASM web sites `http://www.eecs.umich.edu/gasm` and `http://www.uni-paderborn.de/cs/asm.html`.

References

[1] W. Ahrendt. Von Prolog zur WAM. Verifikation der Prozedurübersetzung mit KIV. Master's thesis, Universität Karlsruhe, Karlsruhe, Germany, 1995.

In German, starting point for [113]. See comment to [49].

[2] M. Anlauff, P. Kutter, and A. Pierantonio. Formal Aspects of and Development Environments for Montages. In M. Sellink, editor, *2nd International Workshop on the Theory and Practice of Algebraic Specifications*, Workshops in Computing, Amsterdam, 1997. Springer.

A description of the use of Montages [90] and the Gem-Mex tool, with some small examples.

[1] To appear in *Formal Specification Column* (Ed. H.Ehrig), Bulletin of the EATCS 64, February 1998.

[3] L. Araujo. Correctness proof of a Distributed Implementation of Prolog by means of Abstract State Machines. *Journal of Universal Computer Science*, 3(5):416–422, 1997.

Building upon [49], 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.

[4] D. Bèauquier and A. Slissenko. On Semantics of Algorithms with Continuous Time. Technical Report 97-15, Dept. of Informatics, Université Paris-12, October 1997.

A continuation of [5]. The authors consider a class of algorithms with explicit continuous time (a modified version of ASMs), a logic which suffices to write requirements specifications close to natural language, and the corresponding verification problem, all in a single logic. An enhanced logic from that used in [5] is presented and used to give a proof of correctness of the Railroad Crossing problem [78].

[5] D. Bèauquier 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, editors, *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE*, volume 1214 of *LNCS*, pages 201–212. Springer, 1997.

The ASM specification of the railroad crossing problem [78] is analyzed to create an appropriate timed-transition system, suitable for algorithmic model checking. An early version appeared in 1995 as Technical Report 96-10 of Dept. of Informatics, Université Paris-12. For a continuation see [4].

[6] B. Beckert and J. Posegga. leanEA: A Lean Evolving Algebra Compiler. In H. Kleine Büning, editor, Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL'95), volume 1092 of *LNCS*, pages 64–85. Springer, 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.

[7] C. Beierle. Formal Design of an Abstract Machine for Constraint Logic Programming. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 377–382, Elsevier, Amsterdam, the Netherlands, 1994.

Proposes a general implementation scheme for CLP(X) over an unspecified constraint domain X by designing a generic extension WAM(X) of the Warren Abstract Machine and a corresponding generic compliation 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 [49].

[8] C. Beierle and E. Börger. Correctness Proof for the WAM With Types. In E. Börger, G. Jäger, H. Kleine Büning, and M. M. Richter, editors, *Computer Science Logic*, volume 626 of *LNCS*, pages 15–34. Springer, 1992.

The Börger-Rosenzweig specification and correctness proof for compiling Prolog to WAM [49] 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 constraint) 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. [9] 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 [10].

[9] 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 [10] 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.

[10] 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 [8].

[11] C. Beierle, E. Börger, I. Durdanovic, 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, editors, *Formal Methods for Industrial Applications. Specifying and Programming the Steam-Boiler Control*, number 1165 in LNCS, pages 62–78. Springer, 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-Meeting 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 line of research see [37].

[12] G. Bella and E. Riccobene. Formal Analysis of the Kerberos Authentication System. *Journal of Universal Computer Science*, 3(12), 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.

[13] B. Blakley. *A Smalltalk Evolving Algebra and its Uses*. PhD thesis, University of Michigan, Ann Arbor, Michigan, 1992.

An early student work on ASMs (the late date of 1992 is accidental). A reduced version of Smalltalk is formalized and studied.

[14] A. Blass and Y. Gurevich. The Linear Time Hierarchy Theorems for Abstract State Machines. *Journal of Universal Computer Science*, 3(4):247–278, 1997.

Contrary to polynomial time, linear time badly depends on the computation model. In 1992, Neil Jones designed a couple of computation models where the linear-speed-up theorem fails and linear-time computable functions form a proper hierarchy. However, the linear time of Jones' models is too restrictive. Linear-time hierarchy theorems for random access machines and ASMs are proven. In particular it is shown that there exists a sequential ASM U (an allusion to "universal") and a constant c such that, under honest time counting, U simulates every other sequential ASM in lock-step with log factor c. The generalization for ASMs is harder and more important because of the greater flexibility of the ASM model. One long-term goal of this line or research is to prove linear lower bounds for linear time problems. The result has been anounced unter 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.

[15] A. Blass, Y. Gurevich, and S. Shelah. Choiceless Polynomial Time. Technical Report CSE-TR-338-97, EECS Dept., University of Michigan, 1997.

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 [71]. 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.

[16] 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, editors, *CSL'89. 3rd Workshop on Computer Science Logic*, volume 440 of *LNCS*, pages 36–64. Springer, 1990.

See Comments to [18].

[17] E. Börger. A Logical Operational Semantics of Full Prolog. Part II: Built-in Predicates for Database Manipulation. In B. Rovan, editor, *Mathematical Foundations of Computer Science*, volume 452 of *LNCS*, pages 1–14. Springer, 1990.

See Comments to [18].

[18] E. Börger. A Logical Operational Semantics for Full Prolog. Part III: Built-in Predicates for Files, Terms, Arithmetic and Input-Output. In Y. Moschovakis, editor, *Logic From Computer Science*, volume 21 of *Berkeley Mathematical Sciences Research Institute Publications*, pages 17–50. Springer, 1992.

This paper, along with [16] and [17] are the original 3 papers of Börger where he gives a complete ASM formalization of Prolog with all features discussed in the international Prolog standardization working group (WG17 of ISO/IEC JTCI SC22), see [23]. The specification is developed by stepwise refinement, describing orthogonal language features by modular rule sets. An improved (tree instead of stack based) version is found in [43, 48]; the revised final version is in [48]. These three papers were also published in 1990 as IBM Germany Science Center Research Reports 111, 115 and 117 respectively. The refinement technique is further developed in [49, 27, 36, 37, 54].

[19] E. Börger. Logic Programming: The Evolving Algebra Approach. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 391–395, Elsevier, Amsterdam, the Netherlands, 1994.

Surveys the work which has been done from 1986–1994 on specifications of logic programming systems by ASMs.

[20] E. Börger. Annotated Bibliography on Evolving Algebras. In E. Börger, editor, *Specification and Validation Methods*, pages 37–51. Oxford University Press, 1995.

An annotated bibliography of papers (as of 1994) which deal with or use ASMs.

[21] E. Börger. Why Use Evolving Algebras for Hardware and Software Engineering? In M. Bartosek, J. Staudek, and J. Wiederman, editors, *Proceedings of SOFSEM'95, 22nd Seminar on Current Trends in Theory and Practice of Informatics*, volume 1012 of *LNCS*, pages 236–271. Springer, 1995.

A presentation of the salient features of ASMs, as part of a discussion and survey of the use of ASMs in design and analysis of hardware and software systems. The leading example is elaborated in detail in [36].

[22] E. Börger. Evolving Algebras and Parnas Tables. In H. Ehrig, F. von Henke, J. Meseguer, and M. Wirsing, editors, *Specification and Semantics*. Dagstuhl Seminar No. 9626, July 1996.

Extended abstract showing that Parnas' approach to use function tables for precise program documentation can be generalized and gentilized in a natural way by using ASMs for well-documented program development.

[23] E. Börger and K. Dässler. Prolog: DIN Papers for Discussion. ISO/IEC JTCI SC22 WG17 Prolog Standardization Document 58, National Physical Laboratory, Middlesex, England, 1990.

A version of [16, 17, 18] proposed to the International Prolog Standardization Committee as a complete formal semantics of Prolog. An improved version is in [48].

[24] 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, editor, *Proceedings of the First IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'95)*, pages 145–148, November 1995.

Presents a technique, based on ASMs, by which a behavioural description of a processor is obtained as result of the composition of its (formally specified) basic architectural components. The technique is illustrated on the example of a subset the zCPU processor (used as 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.

[25] E. Börger, G. Del Castillo, P. Glavan, and D. Rosenzweig. Towards a Mathematical Specification of the APE100 Architecture: the APESE Model. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 396–401, Elsevier, Amsterdam, the Netherlands, 1994.

Defines an ASM model of the high-level programmer's view of the APE100 parallel architecture. This simple model is refined in [24] to an ASM processor model.

[26] E. Börger and B. Demoen. A Framework to Specify Database Update Views for Prolog. In M. J. Maluszynski, editor, *PLILP'91. Third International Symposium on Programming Languages Implementation and Logic Programming.*, volume 528 of *LNCS*, pages 147–158. Springer, 1991.

Provides a precise definition of the major Prolog database update views (immediate, logical, minimal, maximal), within a framework closely related to [16, 17, 18]. A preliminary version of this was published as *The View on Database Updates in Standard Prolog: A Proposal and a Rationale* in ISO/ETC JTCI SC22 WG17 Prolog Standardization Report no. 74, February 1991, pp 3-10.

[27] 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 [33]. Sharpens the refinement method of [49] 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 [28], 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 [24, 36]).

[28] E. Börger, I. Durdanović, and D. Rosenzweig. Occam: Specification and Compiler Correctness. Part I: Simple Mathematical Interpreters. In U. Montanari and E. R. Olderog, editors, *Proc. PROCOMET'94 (IFIP Working Conference on Programming Concepts, Methods and Calculi)*, pages 489–508. North-Holland, 1994.

A truly concurrent ASM model of Occam is defined as basis for a provably correct, smooth transition to the Transputer Instruction Set architecture. This model is stepwise refined, in a provably 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 [27] for the extension of this work to cover the compilation to Transputer code.

[29] E. Börger and U. Glässer. A Formal Specification of the PVM Architecture. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 402–409, Elsevier, Amsterdam, the Netherlands, 1994.

Provides an ASM model for the Parallel Virtual machine (PVM, the Oak Ridge National Laboratory software system that serves as a general purpose environment for heterogeneous distributed computing). The model 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 International Conference on Parallel and Distributed Computing Systems* (PDCS'94), Las Vegas/Nevada, 5.-9.10.1994. See [30] for an elaboration of this paper.

[30] E. Börger and U. Glässer. Modelling and Analysis of Distributed and Reactive Systems using Evolving Algebras. In Y. Gurevich and E. Börger, editors, *Evolving Algebras – Mini-Course, BRICS Technical Report (BRICS-NS-95-4)*, pages 128–153. University of Aarhus, Denmark, July 1995.

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

[31] 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*, pages 500–505, Los Alamitos, California, 1994. IEEE CS 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,inertial*, and the three process suspensions (wait on/until/for). Shared variables, postponed processes and rejection pulse are covered. The work is extended in [32].

[32] 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, editors, *Formal Semantics for VHDL*, pages 107–139. Kluwer Academic Publishers, 1995.

Extends the work in [31] by including the treatment of variable assignments and of value propagation by ports. This ASM model for VHDL is extended to analog VHDL in [111].

[33] E. Börger and Y. Gurevich. Evolving Algebras – Mini Course. In E. Börger and Y. Gurevich, editors, *BRICS Technical Report (BRICS-NS-95-4)*, pages 195–222. University of Aarhus, 1995.

Contains reprints of the papers [14, 72, 73, 75, 77, 79, 76, 34, 24, 27, 30].

[34] E. Börger, Y. Gurevich, and D. Rosenzweig. The Bakery Algorithm: Yet Another Specification and Verification. In E. Börger, editor, *Specification and Validation Methods*, pages 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, mutatis mutandis, for durative actions.

[35] 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, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 410–415, 1994.

Defines an ASM model for the innermost version of the functional logic programming language BABEL, extending the Prolog model of [48] 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.

[36] E. Börger and S. Mazzanti. A Practical Method for Rigorously Controllable Hardware Design. In J.P. Bowen, M.B. Hinchey, and D. Till, editors, *ZUM'97: The Z Formal Specification Notation*, volume 1212 of *LNCS*, pages 151–187. Springer, 1996.

A technique for specifying and verifying the control of pipelined microprocessors is described, illustrated through formal models for Hennessy and Patterson's RISC architecture DLX. A sequential DLX model is stepwise refined to the pipelined DLX 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, ATT Bell Laboratories, Bellcore) research report TR 96-22, pp.1-60, Brunswick, New Jersey, 1995.

[37] E. Börger and L. Mearelli. Integrating ASMs into the Software Development Life Cycle. *Journal of 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 smoothly 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" (May 1997). Source code (the ultimate refinement) for the case study appears in [95]; the model checking results for the ASM models appears in [124].

[38] E. Börger and E. Riccobene. Logical Operational Semantics of Parlog. Part I: And-Parallelism. In H. Boley and M. M. Richter, editors, *Processing Declarative Knowledge*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 191–198. Springer, 1991.

See comment to [41].

[39] 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 [41].

[40] E. Börger and E. Riccobene. Logical Operational Semantics of Parlog. Part II: Or-Parallelism. In A. Voronkov, editor, *Logic Programming*, volume 592 of *Lecture Notes in Artificial Intelligence*, pages 27–34. Springer, 1992.

See comment to [41].

[41] E. Börger and E. Riccobene. A Formal Specification of Parlog. In M. Droste and Y. Gurevich, editors, *Semantics of Programming Languages and Model Theory*, pages 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. Improved and extended version of [38, 40], obtained combining the concurrent features of the Occam model of [81] with the logic programming model of [48]. Also published as Technical Report TR 1/93 from Dipartmento di Informatica, Università da Pisa, 1993.

[42] E. Börger and E. Riccobene. Logic + Control Revisited: An Abstract Interpreter for Gödel Programs. In G. Levi, editor, *Advances in Logic Programming Theory*. Oxford University Press, 1994.

Develops a simple ASM interpreter for Gödel programs. This interpreter abstracts from the deterministic and sequential execution strategies of Prolog [49] 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.

[43] E. Börger and D. Rosenzweig. A Formal Specification of Prolog by Tree Algebras. In V. Čeric, V. Dobrić, V. Lužar, and R. Paul, editors, *Information Technology Interfaces*, pages 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 to replace the stack based model of [16] and the stack implementation of the tree based model of [17] by a pure tree model for Prolog. An improved version of the latter is the basis for [48] where also an error in the treatment of the *catch* built-in predicate is corrected.

[44] E. Börger and D. Rosenzweig. An Analysis of Prolog Database Views and their Uniform Implementation. Research Report CSE-TR-89-91, EECS Dept., University of Michigan, Ann Arbor, Michigan, 1991.

A mathematical analysis of the Prolog database views defined in [26]. The analysis is derived by stepwise refinement of the stack model for Prolog from [49]. 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 by the international Prolog Standardization Committee as ISO/IEC JTCI SC22 WG17 document no. 80, National Physical Laboratory, Teddington, England 1991.

[45] 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, editors, *CSL'90, 4th Workshop on Computer Science Logic*, volume 533 of *LNCS*, pages 31–66. Springer, 1991.

Refines Börger's Prolog model [17] 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 [46] and improved in [49].

[46] E. Börger and D. Rosenzweig. WAM Algebras – A Mathematical Study of Implementation, Part 2. In A. Voronkov, editor, *Logic Programming*, volume 592 of *Lecture Notes in Artificial Intelligence*, pages 35–54. Springer, 1992.

Refines the Prolog model of [45] 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 [49]. Also issued as Technical Report CSE-TR-88-91 from EECS Dept, University of Michigan, Ann Arbor, Michigan, 1991.

[47] E. Börger and D. Rosenzweig. The Mathematics of Set Predicates in Prolog. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory*, volume 713 of *LNCS*, pages 1–13. Springer, 1993.

Provides a logical (proof–theoretical) specification of the solution collecting predicates *findall, bagof* of Prolog. This abstract 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 section 5, on solution collecting predicates, in [48]. Also issued as *Prolog. Copenhagen papers 2*, ISO/IEC JTC1 SC22 WG17 Standardization report no. 105, National Physical Laboratory, Middlesex, 1993, pp. 33-42.

[48] E. Börger and D. Rosenzweig. A Mathematical Definition of Full Prolog. In *Science of Computer Programming*, volume 24, pages 249–286. North-Holland, 1994.

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 [16, 17, 23, 43]. An abstract of this was issued as *Full Prolog in a Nutshell* in *Logic Programming* (Proceedings of the 10th International Conference 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.

[49] E. Börger and D. Rosenzweig. The WAM – Definition and Compiler Correctness. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, Studies in Computer Science and Artificial Intelligence, chapter 2, pages 20–90. North-Holland, 1994.

Substantial example of the successive refinement method in the area, improving [16, 17, 18] and the direct predecessors [45, 46]. A hierarchy of ASMs provides a solid foundation for constructing provably correct compilers from Prolog to WAM. Various refinement steps take care of different distinctive features ("orthogonal components" in the authors' metaphor) of WAM making the specification as well as the correctness proof modular and extendible; examples of such extensions are found in [9, 10, 50, 3, 92]. An extension of this work to an imperative language with parallelism and non determinism has been provided in [27]. See [1, 107, 113] for machine checked versions of the correctness proofs (for some of) the refinement steps. A preliminary version appeared as Research Report TR-14/92, Dipartimento di Informatica, Università di Pisa, 1992.

[50] E. Börger and R. Salamone. CLAM Specification for Provably Correct Compilation of CLP($\mathcal{R}$) Programs. In E. Börger, editor, *Specification and Validation Methods*, pages 97–130. Oxford University Press, 1995.

Extends the Börger–Rosenzweig's specification and correctness proof, for compiling Prolog programs to the WAM [49], 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)", Master's Thesis, Università di Pisa, Italy, 1993.

[51] 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, editors, *CSL'90, 4th Workshop on Computer Science Logic*, volume 533 of *LNCS*, pages 67–79. Springer, 1991.

An ASM formalization of Alain Colmerauer's constraint logic programming language Prolog III, obtained from the Prolog model in [16, 17, 18] through extending unifications by constraint systems. This extension was the starting point for the extension of [49] in [8]. A preliminary version of this was issued as IBM Germany IWBS Report 144, 1990.

[52] 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, the authors give an operational description of the tableau method in terms of ASMs at various levels of refinement ending after four stages at a specification that is very close to the leanT^AP implementation of the tableau calculus in Prolog. Proofs of correctness and completeness of the refinement steps are given.

[53] E. Börger and W. Schulte. A Modular Design for the Java VM architecture. In E. Börger, editor, *Architecture Design and Validation Methods*. Springer, 1998.

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 [54]. The ASM model for JVM defined here and the ASM model for Java defined in [54] 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.

[54] E. Börger and W. Schulte. Programmer Friendly Modular Definition of the Semantics of Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, LNCS. Springer, 1998.

Provides a system and machine independent definition of the semantics of the full programming language Java as it is 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. The definition is intended as basis for the standardization of the semantics of the Java language and of its implementation on the Java Virtual Machine, see the ASM model for the Java VM in [53]. An extended abstract has been presented to the IFIP WG 2.2 (University of Graz, 22.-26.9.1997) by E.Börger and under the title *Modular Dynamic Semantics of Java* to the Workshop on Programming Languages (Ahrensdorp, FEHMARN island, September 25, 1997) by W. Schulte, see University of Kiel, Dept. of CS Research Report Series, TR *Arbeitstagung Programmiersprachen* 1997.

[55] G. Del Castillo, I. Durdanović, and U. Glässer. An Evolving Algebra Abstract Machine. In H. Kleine Büning, editor, Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL'95), volume 1092 of *LNCS*, pages 191–214. Springer, 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. A preliminary version appeared under the title *Specification and Design of the EAM (EAM - Evolving Algebra Abstract Machine)* as Technical Teport tr-rsfb-96-003, Paderborn University, 1996.

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

A demonstration that, in a strong sense, Schoenhage'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.

[57] S. Diehl. Transformations of Evolving Algebras. In *Proceedings of LIRA'97 (VIII International Conference on Logic and Computer Science)*, pages 43–50, Novi Sad, Yugoslavia, September 1997.

First, constant propagation is defined as a transformation on ASMs. Then ASMs are extended by macro definitions and folding and unfolding transformations for macros are defined. Next a simple transformation to flatten transition rules is introduced. Finally a pass separation transformation for ASMs is presented. 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. Next pass separation is applied to a simple interpreter. Finally a comparison to other work is given. A preliminary version appeared in 1995 as Technical Report 02/95 of Universität des Saarlandes.

[58] D. Diesen. *Specifying Algorithms Using Evolving Algebra. Implementation of Functional Programming Languages*. Dr. scient. degree 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 proposed extension to the language of ASMs.

[59] B. Fordham, S. Abiteboul, and Y. Yesha. Evolving Databases: An Application to Electronic Commerce. In *Proceedings of the Interational Database Engineering and Applications Symposium (IDEAS)*, August 1997.

The authors present a rich and extensible database model called "evolving databases" (EDB), with a rich and precise semantics based on ASMs. The authors apply EDBs to electronic commerce applications.

[60] M. Gaieb. Géneration de spécifications Centaur á partir de specifications Montages. Master's thesis, Université de Nice – Sophia Antipolis, June 1997.

This works investigate the possibilities of mapping the operational ASM semantics of the static analysis phase of Montages [90] 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 [90]. In addition, the Gem-Mex Montages tool is interfaced to the Centaur system (which executes Natural Semantics specificaions), and the tool suport of Centaur is exploited in order to generate structural editors for languages defined with Montages.

[61] T. Gaul. An Abstract State Machine specification of the DEC-Alpha Processor Family. Verifix Working Paper [Verifix/UKA/4], University of Karlsruhe, 1995.

An ASM for the DEC-Alpha processor family, derived directly from the original manufacturer's handbook. The specification omits certain less-used instructions and VAX compatibility parts.

[62] U. Glässer. Systems Level Specification and Modelling of Reactive Systems: Concepts, Methods, and Tools. In R. Moreno Diaz F. Pichler and R. Albrecht, editors, *Computer Aided Systems Theory–EUROCAST'95:* Proc. of the Fifth International Workshop on Computer Aided Systems Theory (Innsbruck, Austria, May 1995*)*, volume 1030 of *LNCS*, pages 375–385. Springer, 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.

[63] U. Glässer. Combining Abstract State Machines with Predicate Transition Nets. In F. Pichler and R. Moreno-Díaz, editors, *Computer Aided Systems Theory–EUROCAST'97 (Proc. of the 6th International Workshop on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, Feb. 1997)*, volume 1333 of *LNCS*, pages 108–122. Springer, 1997.

The work investigates the formal relation between ASMs and Pr/TPredicate Transition (Pr/T-) Nets with the aim to integrate 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.

[64] U. Glässer and R. Karges. Abstract State Machine Semantics of SDL. *Journal of Universal Computer Science*, 3(12), 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.

[65] P. Glavan and D. Rosenzweig. Communicating Evolving Algebras. In E. Börger, H. Kleine Büning, G. Jäger, S. Martini, and M. M. Richter, editors, *Computer Science Logic*, volume 702 of *Lecture Notes in Computer Science*, pages 182–215. Springer, 1993.

A theory of concurrent computation within the framework of ASMs is developed, generalizing [81, 38, 40, 39]. As illustration models are given for the Chemical Abstract Machine and the π-calculus. See [75] for a more satisfactory definition of the notion of distributed ASM runs.

[66] P. Glavan and D. Rosenzweig. Evolving Algebra Model of Programming Language Semantics. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 416–422, Elsevier, Amsterdam, the Netherlands, 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.

[67] 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, editors, *Next Generation Information Technology*, volume 504 of *LNCS*, pages 144–160. Springer, 1991.

Uses ASMs to define 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).

[68] E. Grädel and Y. Gurevich. Metafinite Model Theory. In *Logic and Computational Complexity, Selected Papers*, number 960 in LNCS, pages 313–366. Springer, 1995.

A closer look reveals that 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 are defined and the approach and methods of finite model theory are extended to metafinite models. The relevance to the ASM methodology: ASM states are metafinite structures. 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 *Journal of Symbolic Logic*. A revised version is going to appear in 1998 in a special issue of *Information and Computation*.

[69] R. Groenboom and G. Renardel de Lavalette. A Formalization of Evolving Algebras. In *Proceedings of Accolade95*. Dutch Research School in Logic, 1995.

The authors present the syntax and semantics for a Formal Language for Evolving Algebra (FLEA). This language is then extended to a multi-modal language FLEA' and it is sketched how we can transfer the axioms of the logic MLCM to FLEA'. MLCM is a Modal Logic of Creation and Modification based on QDL as presented by Harel.

[70] Y. Gurevich. Algorithms in the World of Bounded Resources. In R. Herken, editor, *The Universal Turing Machine – A Half-Century Story*, pages 407–416. Oxford University Press, 1988.

Early complexity theoretical motivation for the introduction of ASMs is discussed.

[71] Y. Gurevich. Logic and the Challenge of Computer Science. In E. Börger, editor, *Current Trends in Theoretical Computer Science*, pages 1–57. Computer Science Press, 1988.

The introduction and the first use of ASMs (at the end of the paper).

[72] Y. Gurevich. Evolving Algebras. A Tutorial Introduction. *Bulletin of EATCS*, 43:264–284, 1991.

The first tutorial on ASMs. The ASM thesis is stated: Every algorithm can be simulated by an appropriate ASM in lock-step on the natural abstraction level of the algorithm. A slightly revised version of this was reprinted in G. Rozenberg and A. Salomaa Eds, *Current Trends in Theoretical Computer Science*, World Scientific, 1993, pp 266-292. For a more advanced definition see [75].

[73] Y. Gurevich. Evolving Algebras. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 423–427, Elsevier, Amsterdam, the Netherlands, 1994.

The opening talk at the first ASM workshop. Sections: Introduction, The ASM Thesis, Remarks, Future Work.

[74] Y. Gurevich. Logic Activities in Europe. *ACM SIGACT News*, 1994.

A critical analysis of European logic activities in computer science. The part relevant to ASMs is subsection 4.6 called Mathematics and Pedantics.

[75] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

The tutorial [72] covered basic ASMs. In the meantime, ASMs have been extensively used, in particular, for specifying parallel, distributed computations and computations involving real time. It became obvious that a more advanced definition of ASMs is needed. The guide addresses this need. For a recent update *May 1997 Draft of the ASM Guide* see the Technical Report CSE-TR-336-97, EECS Dept., University of Michigan.

[76] 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, editors, *Computer Science Logic*, volume 702 of *LNCS*, pages 274–309. Springer, 1993.

The method of successive refinements (inspired by its application in [16, 17]) is used to give a succint dynamic semantics of the C programming language. For a correction of minor errors and omissions see the ERRATA in LNCS 832 (1994), 334-336. 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, University of Michigan, Ann Arbor, 1995. For an extension to C++ see [121].

[77] Y. Gurevich and J. Huggins. Evolving Algebras and Partial Evaluation. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 587–592, Elsevier, Amsterdam, the Netherlands, 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.

[78] Y. Gurevich and J. Huggins. The Railroad Crossing Problem: An Experiment with Instantaneous Actions and Immediate Reactions. In *Proceedings of CSL'95 (Computer Science Logic)*, volume 1092 of *LNCS*, pages 266–290. Springer, 1996.

An ASM solution for the railroad crossing problem. The paper experiments with agents that perform instantaneous actions in continuous time and in particular with agents that fire 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 relation to model checking see [5, 4].

[79] 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.

[80] Y. Gurevich and R. Mani. Group Membership Protocol: Specification and Verification. In E. Börger, editor, *Specification and Validation Methods*, pages 295–328. Oxford University Press, 1995.

An interesting and useful protocol of Flavio Cristian involves timing constraints and its correctness is not obvious. The protocol is formally specified and verified. (The verification proof allowed the authors to simplify the assumptions slightly.).

[81] Y. Gurevich and L. Moss. Algebraic Operational Semantics and Occam. In E. Börger, H. Kleine Büning, and M. M. Richter, editors, *CSL'89, 3rd Workshop on Computer Science Logic*, volume 440 of *LNCS*, pages 176–192. Springer, 1990.

The first application of ASMs to distributed parallel computing with the challenge of true concurrency. See [28, 27].

[82] Y. Gurevich, N. Soparkar, and C. Wallace. Formalizing Database Recovery. *Journal of 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 each model. An updated version of [123] and of the Technical Reports CSE-TR-249-95 and CSE-TR-327-97 of EECS Department, University of Michigan, Ann Arbor.

[83] Y. Gurevich and M. Spielmann. Recursive Abstract State Machines. *Journal of Universal Computer Science*, 3(4):233–246, 1997.

The authors suggest a definition of recursive ASMs in terms of distributed ASMs. Preliminary version appeared as Technical Report CSE-TR-322-96, EECS Department, University of Michigan, Ann Arbor, 1996.

[84] J. Huggins. Kermit: Specification and Verification. In E. Börger, editor, *Specification and Validation Methods*, pages 247–293. Oxford University Press, 1995.

The Kermit file-transfer protocol (including a sliding windows extension to the basic protocol) is specified and verified using ASMs at several different layers of abstraction. 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.

[85] J. Huggins. Broy-Lamport Specification Problem: A Gurevich Abstract State Machine Solution. Technical Report CSE-TR-320-96, EECS Dept., University of Michigan, 1996.

An ASM solution to a specification problem suggested by Manfred Broy and Leslie Lamport, in conjunction with the Dagstuhl Workshop on Reactive Systems, held in Dagstuhl, Germany, 26-30 September, 1994. Preliminary version appeared as Technical Report CSE-TR-223-94, EECS Department, University of Michigan, Ann Arbor, 1994.

[86] J. Huggins and D. Van Campenhout. Specification and Verification of Pipelining in the ARM2 RISC Microprocessor. Technical Report CSE-TR-321-96, EECS Dept., University of Michigan, 1996.

A layered ASM specification of the ARM2, one of the early commerical RISC microprocessors. The layered specification is used to prove the correctness of the ARM2's pipelining techniques. Extended abstract appears in *Proceedings of the IEEE International High Level Design Validation and Test Workshop (HLDTV'97)*, November 1997.

[87] D. Johnson and L. 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 [81, 41] and from the dynamic tree generation appearing in [43, 48]. Then the simple model of context free rules is extended to characterise in a uniform and natural way different context sensitive languages in terms of ASMs. See [99, 100].

[88] A. Kaplan and J. Wileden. Formalization and Application of a Unifying Model for Name Management. In *The Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, volume 20(4) of *Software Engineering Notes*, pages 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.

[89] A. M. Kappel. Executable Specifications Based on Dynamic Algebras. In A. Voronkov, editor, *Logic Programming and Automated Reasoning*, volume 698 of *Lecture Notes in Artificial Intelligence*, pages 229–240. Springer, 1993.

Defines a language for specification of ASMs and designs an abstract target machine (namely a Prolog program) which is specially tailored for executing ASM computations. A prototype of the compiler has been implemented in Prolog. For a full version see A. M. Kappel, "Implementation of Dynamic Algebras with an Application to Prolog", Master's Thesis, Universität Dortmund, Germany, 1990.

[90] P. Kutter and A. Pierantonio. Montages: Specifications of Realistic Programming Languages. *Journal of 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 Universita de L'Aquila.

[91] P. Kutter and A. Pierantonio. The Formal Specification of Oberon. *Journal of Universal Computer Science*, 3(5):443–503, 1997.

A presentation of the syntax, static semantics, and dynamic semantics of Oberon, using ASMs and Montages [90]. The dynamic semantics previously appeared as P. Kutter, "Dynamic Semantics of the Oberon Programming Language", TIK-Report 25, ETH-Zürich, Feburary 1997.

[92] K. Kwon. A Structured Presenation of a Closure-Based Compilation Method for a Scoping Notion in Logic Programming. *Journal of 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 [49].

[93] A. Lisitsa and G. Osipov. Evolving algebras and labelled deductive systems for the semantic network based reasoning. In *Proceedings of the Workshop on Applied Semiotics, ECAI'96*, pages 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.

[94] W. May. Specifying Complex and Structured Systems with Evolving Algebras. In *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE*, number 1214 in LNCS, pages 535–549. Springer, 1997.

An approach is presented for specifying complex, structured systems with ASMs by means of aggregation and composition.

[95] L. Mearelli. Refining an ASM Specification of the Production Cell to C^{++} Code. *Journal of Universal Computer Science*, 3(5):666–688, 1997.

Source code for the specification problem described in [37].

[96] 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.

[97] J. Morris. *Algebraic Operational Semantics and Modula-2.* PhD thesis, University of Michigan, Ann Arbor, Michigan, 1988.

The earliest formalization of a real-life language. The semantical description is parse-tree directed. In the meantime, the methodology has developed enabling more elegant descriptions, but one has to start somewhere. A PhD thesis under the supervision of Yuri Gurevich. 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*, Springer LNCS 329, 1988, pp. 81-101.

[98] J. Morris and G. Pottinger. Ada-Ariel Semantics. Odyssey Research Associates, Manuscript, July 1990.

[99] L. S. Moss and D. E. Johnson. Dynamic Interpretations of Constraint-Based Grammar Formalisms. *Journal of Logic, Language, and Information*, 4(1):61–79, 1995.

Extends the work of [87] to grammar formalisms based on Kasper-Rounds logics.

[100] L. S. Moss and D. E. Johnson. Evolving Algebras and Mathematical Models of Language. In L. Polos and M. Masuch, editors, Applied Logic: How, What, and Why, volume 626 of *Synthese Library*, pages 143–175. Kluwer Academic Publishers, 1995.

Extends the work of [87] to several other grammar formalisms.

[101] B. Müller. A Semantics for Hybrid Object–Oriented Prolog Systems. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, Elsevier, Amsterdam, the Netherlands, 1994.

Extends the rules given in [48] 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.

[102] 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 w.r.t. programming language semantics given in the framework of ASMs.

[103] A. Poetzsch-Heffter. Comparing Action Semantics and Evolving Algebra based Specifications with respect to Applications. In *Proceedings of the First International Workshop on Action Semantics*, 1994.

Action semantics is compared to ASM based language specifications. In particular, different aspects relevant to language documentation and programming tool development are discussed.

[104] A. Poetzsch-Heffter. Deriving Partial Correctness Logics From Evolving Algebras. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 434–439, Elsevier, Amsterdam, the Netherlands, 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.

[105] A. Poetzsch-Heffter. Developing Efficient Interpreters based on Formal Language Specifications. In P. Fritzson, editor, *Compiler Construction*, volume 786 of *LNCS*, pages 233–247. Springer, 1994.

Reports on extensions of the MAX system enabling the generation and refinement of interpreters based on formal language specifications. In these specifications, static semantics is defined by an attribution mechanism and dynamic semantics is defined by ASMs. Included in [106].

[106] 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. 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. This work is partly based upon [105].

[107] C. Pusch. Verification of compiler correctness for the WAM. In J.Harrison J. von Wright, J.Grundy, editor, *Theorem Proving in Higher Order Logics (TPHOLs'96)*, volume 1125 of *LNCS*, pages 347–362. Springer, 1996.

See comment to [49].

[108] H. Reichel. Unifying ADT and Evolving Algebra Specifications. *Bulletin of EATCS*, 59:112–126, 1996.

[109] E. Riccobene. *Modelli Matematici per Linguaggi Logici*. PhD thesis, University of Catania, 1992.

In Italian. Systematic treatment of ASM models for Gödel [42], Parlog [41], Concurrent Prolog [39], GHC, Pandora.

[110] D. Rosenzweig. Distributed Computations: Evolving Algebra Approach. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 440–441, Elsevier, Amsterdam, the Netherlands, 1994.

Remarks on some ASM models of concurrent and parallel computation.

[111] H. Sasaki, K. Mizushima, and T. Sasaki. Semantic Validation of VHDL-AMS by an Abstract State Machine. In *Proceedings of BMAS'97 (IEEE/VIUF International Workshop on Behavioral Modeling and Simulation)*, pages 61–68, Arlington, VA, October 20-21 1997.

The paper extends the ASM model defined for VHDL in [31, 32], to provide a rigorous definition of VHDL-AMS following the IEEE Language Reference Manual for the analogue extension of VHDL. Reflecting the analysis made in this paper on the BREAK statement, 1076.1 WG will update the LRM draft (e-mail from Dr. Hisashi Sasaki, Mixed Signal Design Automation Sec., Analog and Mixed Signal LSI Design Dept., TOSHIBA CORPORATION, Japan).

[112] J. Sauer. *Wissensbasiertes Lösen von Ablaufsplanungsproblemen durch explizite Heuristiken*. PhD thesis, Universität Oldenburg, 1993.

Published in: Dissertationen zur Künstlichen Intelligenz, Infix-Verlag, Dr. Ekkehardt Hundt, St. Augustin 1993, pp. 204. Uses ASMs to define the semantics for selection and elaboration of heuristics for computation of goal sets in the language HERA. 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.

[113] G. Schellhorn and W. Ahrendt. Reasoning about Abstract State Machines: The WAM Case Study. *Journal of Universal Computer Science*, 3(4):377–413, 1997.

The authors apply the KIV (Karlsruhe Interactive Verifier) system to mechanically verify the proof of correctness of the Prolog to WAM transformation described in [49].

[114] A. Schönegge. Extending Dynamic Logic for Reasoning about Evolving Algebras. Technical Report 49/95, Universität Karlsruhe, Fakultät für Informatik, 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 the KIV (Karlsruhe Interactive Verifier) to reason about ASMs directly.

[115] M. Schrefl and G. Kappel. Cooperation Contracts. In T. J. Teorey, editor, *Proc. 10th International Conference on the Entity Relationship Approach*, pages 285–307. E/R Institute, 1991.

The authors introduce the concept of cooperative message handling and use ASMs to give formal semantics.

[116] K. Stroetmann. The Constrained Shortest Path Problem: A Case Study In Using ASMs. *Journal of Universal Computer Science*, 3(4):304–319, 1997.

An abstract, nondeterministic form of the constrained shortest path problem is defined as an ASM and proven correct, then refined to the level of implementation.

[117] H. Tonino. *A Theory of Many-sorted Evolving Algebras*. Ph.d. thesis, Delft University of Technology, 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.

[118] H. Tonino and J. Visser. Stepwise Refinement of an Anstract State Machine for WHNF-Reduction of λ-Terms. Technical Report 96-154, Faculty of Technical Mathematics and Informatics, Delft University of Technology, 1996.

A series of ASMs for finding the weak head normal form (WHNF) of an arbitrary term of the λ-calculus is presented.

[119] 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 implemented).

[120] J. Visser. Evolving algebras. Master's thesis, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Zuidplantsoen 4, 2628 BZ Delft, The 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.

[121] C. Wallace. The Semantics of the C++ Programming Language. In E. Börger, editor, *Specification and Validation Methods*, pages 131–164. Oxford University Press, 1995.

The semantical description in [76] is extended to encompass all of C++.

[122] 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 [90].

[123] C. Wallace, Y. Gurevich, and N. Soparkar. Formalizing Recovery in Transaction-Oriented Database Systems. In S. Chaudhuri, A. Deshpande, and R. Krishnamurthy, editors, *Proceedings of the Seventh International Conference on Management of Data*, pages 166–185, New Delhi, India, 1995. Tata McGraw-Hill.

The specification and verification of the Undo/Redo algorithm is presented in a discussion of ASMs as a formal tool for database recovery. An early version of [82].

[124] K. Winter. Model Checking for Abstract State Machines. *Journal of Universal Computer Science*, 3(5):689–701, 1997.

A framework is developed for using a model checker to verify ASM models. It is applied to the production cell control model described in [37].

[125] A. Zamulin. Algebraic Specification of Dynamic Objects. In *Proceedings of LMO'97 (Acte du Colloque Langage et Modeles a Objets*, pages 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.

[126] 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, 1997.

A Typed Gurevich Machine [127] is used to define a compiler for Oberon to an algebraically-specified abstract target machine.

[127] A. Zamulin. Typed Gurevich Machines Revisited. Joint CS & IIS Bulletin, Computer Science, 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.

[128] W. Zimmerman and T. Gaul. On the Construction of Correct Compiler Back-Ends: An ASM Approach. *Journal of 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).