2-1998

Abstract State Machines 1988-1998: Commented ASM

Bibliography

James Huggins
Egon Borger

Follow this and additional works at: https://digitalcommons.kettering.edu/computerscience_facultypubs

Part of the Computer Sciences Commons
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

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


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


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


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].


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].


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.


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 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 [10].


The Börger-Rosenzweig specification and correctness proof for compiling Prolog to WAM [19] 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. [8] 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].


Revised version of [8].


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].


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.


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


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 of research is to prove linear lower bounds for linear time problems. 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.
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.

See Comments to [18].

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 JTC1 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].

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

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

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].

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.

A version of [14, 17, 18] proposed to the International Prolog Standardization Committee as a complete formal semantics of Prolog. An improved version is in [15].


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.


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.


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.


The final draft version has been issued in BRICS Technical Report (BRICS-NS-95-4), see [33]. Sharpens the refinement method of [14] 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, 30]).

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.


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.


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.


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].


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 [33].


Contains reprints of the papers [4, 72, 73, 75, 77, 76, 84, 24, 27, 33].


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.


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.


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.


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].


See comment to [41].


An ASM formalization of Ehud Shapiro’s Concurrent Prolog. Adaptation of the model defined for PARLOG in [41].


See comment to [41].


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 Dipartimento di Informatica, Università da Pisa, 1993.


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.


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.


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 [19]. 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 JTC1 SC22 WG17 document no. 80, National Physical Laboratory, Teddington, England 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 [47].


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.


Substantial example of the successive refinement method in the area, improving [16, 17, 18] and the direct predecessors [15, 16]. 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 [4, 11, 50, 14, 22]. An extension of this work to an imperative language with parallelism and non determinism has been provided in [27]. See [4, 17, 11] 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.


Extends the Börger–Rosenzweig’s specification and correctness proof, for compiling Prolog programs to the WAM [43], to CLP(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.


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.


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 leanTAP implementation of the tableau calculus in Prolog. Proofs of correctness and completeness of the refinement steps are given.


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.


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.


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 Report tr-rsfb-96-003, Paderborn University, 1996.


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.


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.


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


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.

This works investigate the possibilities of mapping the operational ASM semantics of the static analysis phase of Montages [30] 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 [30]. 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.


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.


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.


The work investigates the formal relation between ASMs and Pr/T-Predicate 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.


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.


A theory of concurrent computation within the framework of ASMs is developed, generalizing [81, 38]. 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.

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.


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).


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*.


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.


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


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


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].


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


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 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 [4, 5].

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.

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.).

Bünning, and M. M. Richter, editors, CSL’89, 3rd Workshop on Computer Science Logic, volume 440

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


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.


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.


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
of the second author, pp.IX+91, University of Michigan, Ann Arbor, 1995.


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,
of Michigan, Ann Arbor, 1994.

[86] J. Huggins and D. Van Campenhout. Specification and Verification of Pipelining in the ARM2 RISC

A layered ASM specification of the ARM2, one of the early commercial 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 Phi-

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, 82] 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 [83, 100].

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.


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.


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.


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].


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.


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


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

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.


Extends the work of [97] to grammar formalisms based on Kasper-Rounds logics.


Extends the work of [97] to several other grammar formalisms.


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.


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.


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


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.

[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].


[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 [41].
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.

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

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.

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

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).

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

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].


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.


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


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).