Journal of Universal Computer Science, vol. 8, no. 1 (2002), 2-74
submitted: 11/1/02, accepted: 26/1/02, appeared: 28/1/02 © J.UCS

The Origins and the Development of the ASM Method for High Level System Design and Analysis

Egon Börger
(Università di Pisa, Italy
boerger@di.unipi.it)

Abstract: The research belonging to the Abstract State Machines approach to system design and analysis is surveyed and documented in an annotated ASM bibliography. The survey covers the period from 1984, when the idea for the concept of ASMs (under the name dynamic or evolving algebras or structures) appears for the first time in a foundational context, to the year 2001 where a mathematically well-founded, practical system development method based upon the notion of ASMs is in place and ready to be industrially deployed. Some lessons for the future of ASMs are drawn.

Key Words: System design, specification methods, system analysis, abstract state machines, models of computation.
Category: D1,D2,D3,C1,C3,I6,H1,G0,F1.1,F1.2,F3.1,F3.2,F4.2,F4.3

1 Introduction

The ASM method for high-level design and analysis of computing systems naturally grew out of the foundational concern which led to the discovery of the notion of ASMs, although it took some time for the concept to sink in. Indeed as often happens with ideas which change the way we look at things, its "real"ization —through becoming the basis for an intellectual, machine supported instrument for practical system design and analysis—encountered significant resistance in the scientific community. In this paper we survey the already rich ASM literature and the salient steps of the development of the ASM method out of its origins. This reflexion, upon where we came from and upon the way we went, helps to bring into focus some risks and challenges for the future of the ASM approach to industrial system development. We discuss these lessons for the future in the concluding section.

From 1984 to today one can distinguish four phases which we are going to survey in the following sections:

- the start in 1984 with the **idea of an improved Church-Turing thesis** for a "general kind of abstract computational device, called dynamic structures" [175],
- the **recognition of the practical potential** of the abstract machine concept for building and analysing reliable ground models and their controllable

refinements to executable code, an insight which came through the experience gained at the beginning of the 90'ies by extensive modeling of the dynamic semantics of various programming languages and their implementation [49],

- the broadband experimental **practicability test** for the ASM concept in complex real-life applications, an effort which shaped the ASM design and analysis method through numerous modeling and verification projects for real-life architectures, virtual machines, protocols, and control software, carried out in the years 1993-1995 [53] preceding the final definition of ASMs in [181],
- the systematic **integration of the ASM method** into established software development environments which created the practical ASM approach to high-level system design and analysis as one sees it nowadays, ready to be deployed in industrial settings [59], and the completion in 2000/01 of the first part of the original foundational goal, namely by the proof of the sequential ASM thesis from three basic postulates [183] and of its extension to synchronous parallel algorithms [40].

For the references, with few exceptions we stick to list only work which is directly related to ASMs. We revise and complete the annotated bibliography which appeared in [52], was updated in [85] and since then is maintained by Huggins on the ASM website[1]. This paper is based upon and elaborates the historical accounts [55, 57, 60]. It is not an introduction to ASMs, see [181, 59] and for a textbook introduction [291, Ch.2].

2 The Idea of the New Thesis

It was an epistemological concern which led Gurevich to the idea of Abstract State Machines, namely the goal to sharpen the Church-Turing thesis by including the consideration of resource bounds. The problem is stated for the first time in 1984 in a technical report [174], where ASMs appear in embryo under the name of dynamic structures, later also called evolving structures or algebras. A year later the program is formulated in a note to the American Mathematical Society [175] from where we quote the central part:

> First, we adapt Turing's thesis to the case when only devices with bounded resources are considered. Second, we define a more general kind of abstract computational device, called dynamic structures, and put forward the following new thesis: Every computational device can be simulated

[1] http://www.eecs.umich.edu/gasm

> by an appropriate dynamic structure—of appropriately the same size—in real time; a uniform family of computational devices can be uniformly simulated by an appropriate family of dynamic structures in real time. In particular, every sequential computational device can be simulated by an appropriate sequential dynamic structure.

In 1988 a more detailed exposition of this primarily complexity theoretic program appears [177], but again without any concrete definition of the abstract machine concept. Apparently at that time there was still some hesitation on "different classes of dynamic structures" which "may be defined by imposing syntactical restrictions on transition rules, by allowing or forbidding the evolution of the signature (the language) of the current configuration, by allowing or forbidding the creation of new universes (sorts, types) and the elimination of old ones, and so on" (op.cit., pg.413). Into this period falls also the formulation of the Kolmogorov-Uspenskii thesis [221] as stating that "every computation, performing only one restricted local action at a time, can be viewed as the computation of an appropriate Kolmogorov-Uspenskii machine" [178], a subclass of what became known as Schönhage's storage modification machines [283] which later could be characterized as a class of unary sequential ASMs [129].

In a series of lectures on Semantics of Programming Languages, delivered to the computer science PhD program in Pisa in the Spring of 1987, Gurevich explained the concept of ASMs by examples, namely Turing machines, stack machines, and some Pascal programs. Börger learnt ASMs from these lectures and suggested to add the course material to [176] (see sections 10 and 11)[2]. There the proposal appears to use "dynamic structures" for an operational semantics of imperative programming languages, a project tried out for the core of Modula-2 in Morris' PhD thesis [239]. During the winter of 1988/89 ASMs were tried out to define the semantics of Prolog by an execution oriented yet abstract model, which was intended to become complete and precise, but nevertheless of manageable size and reflecting the logical content of Prolog programs in a transparent way, to be useful to programmers [44, 45, 46]. The goal was achieved by introducing the stepwise refinement method into modeling with ASMs, exploiting the possibilities for abstraction that are inherent in the ASM concept. Stepwise refinements allowed to separate orthogonal language features by modules of rule sets (*horizontal refinement*) and to deal with them at different levels of detail (*vertical refinement*), supported by an appropriate classification of functions (which was later refined [53, 59]).

[2] The stack machine and the Turing machine ASMs went into sections 4 and 6 of [179].

3 Recognizing the Practical Relevance of ASMs

Through his sabbatical work at IBM Germany in 1989/1990 Börger realized that his ASM model for Prolog solved some central problems the ISO Prolog standardization committee had faced for years [69], in particular the database update view problem [72, 95] and the semantical problems related to Prolog's solution-collecting predicates [99]. In fact a version of this model became the standard definition of the dynamic semantics of Prolog [211], after many unsuccessful attempts documented in the literature to provide such a definition using traditional approaches (see the detailed discussion in section 4 of [44]). Through the work with the software engineers from IBM, Quintus, Bim, Interface, Siemens, which at the time were developing commercial implementations of Prolog, the usefulness of the ASM concept became apparent for *supporting changing designs* in an industrial standardization and development process. The flexibility was recognized which is gained by using ASMs for modeling and for prototypical (mental or machine supported) simulations. This showed up through the ease with which ASMs allowed the practitioners to perform the following three of their daily duties:

- to rigorously model and document design decisions, **building ground models** [3]in a faithful and objectively checkable manner. This means to turn descriptions expressed in application domain (typically natural language) terms into precise abstract definitions, which the software engineers were comfortable to manipulate as a *semantically well-founded form of pseudo-code over abstract data*,
- to *adapt abstract models to changing requirements*, and to refine them in successive steps—in a controllable and well documentable way—to their implementation, thus providing **practical forms of refinement** for linking ground models by hierarchies of intermediate models (modules) to executable code,
- to turn such precise abstract pseudo-code models into prototypical *executable versions* which can be used for **simulations prior to coding** of the system under development.

Unfolding the potential of the concept of ASMs for such a *method of modeling-for-change*, at the desired level of abstraction, to be used together with an appropriately chosen mathematical verification and experimental validation technique, was the result of extensive experimentation during the years 1990–1992. It was focussed on the following three issues:

[3] The originally chosen term *primary model* [49, Section 3] was replaced later by *ground model* [53].

1. **adaptations and extensions of models** via *horizontal refinements*, realized by refining the basic ASM model for Prolog to some of the major extensions of the language and their implementations, to be precise the following seven ones:
   - Colmerauer's Prolog III, obtained by adding to the unifiability check of the Prolog model a solvability test for general constraints [104],
   - IBM's Protos-L, developed and implemented on the Protos Abstract Machine at IBM Germany, obtained from the Prolog model by adding to it type constraints and a solvability predicate [26, 27, 28],
   - the functional-logical language Babel and its implementation on the Narrowing Machine [87], obtained by adding to the backtracking rules of the Prolog model rules for the reduction of functional expressions to normal form,
   - Lloyd's and Hill's logic programming language Gödel, obtained by abstracting in the Prolog model from the deterministic and sequential execution strategy of ISO Prolog [93],
   - B. Müller's object-oriented Prolog [243, 244], obtained by enriching the four ASM rules for the user–defined core of Prolog [47] with rules for object creation and deletion, data encapsulation, inheritance, messages, polymorphism, and dynamic binding,
   - Sauer's adaption of the Prolog model to an ASM defining the semantics of the domain-specific language HERA, tailored for programming scheduling algorithms for business processes on the basis of given heuristics [273, Ch.3.3],
   - the main parallel extensions of Prolog (see below),
2. **stepwise detailing of models** by *vertical refinements*, realized for the WAM implementation of Prolog by defining a chain of 12 proven to be correct refinement steps which link the high-level Prolog model (in fact its streamlined version in [101]) to its implementation by Warren Abstract Machine code [96, 98, 100]. It turned out that the models and the proofs could be reused and extended to derive the correctness also for the following four implementations:
   - the implementation of a high-level CLP(R) model on the Constrained Logic Arithmetical Machine, developed at IBM Yorktown Heights [102],
   - the implementation of a high-level Protos-L model on the Protos Abstract Machine [27, 28][4],

   - the parallel execution of Prolog on distributed memory [13], see also the related later work [249],
   - the implementation of scoping of procedure definitions in a sublanguage of Lambda-Prolog where implications are allowed in the goals [227],

3. **making abstract models executable** for their experimental validation, realized during the academic year 1989/90 in Kappel's Diplom thesis at the University of Dortmund [219, 220], and a year later at Quintus [112], allowing to simulate the ASM Prolog models defined in [44, 45].

The method of successive refinements of ASMs was applied in [184] to provide a transparent ASM model for the dynamic semantics of C. An earlier version of this work had inspired a similar project for Cobol which was started in [301] (though not continued). Before that, in Blakley's PhD thesis [33] an unstructured ASM model for a subset of Smalltalk had been defined. Inspired by [239], ASMs are used in [166] to provide a succinct operational description of typical object-oriented features like object creation, overriding, dynamic binding, inheritance in the context of data models. In [284] an ASM rule is added to define cooperative message handling, by describing the run-time search of the most specific cooperation contract in the inheritance hierarchy which implements a cooperative message, i.e. a message which involves several objects on the basis of cooperation contracts. Later the modeling of object-oriented programming language features is taken up once more in Ann Arbor, this time using the refinement method to extend the ASM model for C to one for C++ [303].

In 1991 Gurevich writes for his column on Logic in Computer Science in the Bulletin of the EATCS the so-called ASM tutorial [179], which is based on lecture notes from his Fall 1990 course on Principles of Programming Languages at the University of Michigan, containing most notably the first definition of sequential ASMs. The same year a textbook introduction to ASMs is written [48] coming with an illustration by machines which operate on standard data structures and by the tree based core Prolog machine [47], drawn from notes of lectures on Semantics of Programming Languages in a summer school held in 1989 in Cortona/Italy which triggered the first European PhD project on ASMs [264]. Gurevich completed the tutorial definition in the so-called Lipari guide [181], lecture notes of a course delivered in 1993 at the Lipari/Sicily summer school on *Specification and Validation Methods for Programming Languages and Systems* [51]. The definition essentially remained stable since then [5], in fact it constitutes the basis for the proof established five years later for the sequential version of the

[4] Beierle [25] turned this construction into 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.

ASM thesis from three fundamental postulates [183]. As has been observed by Blass [34, Section 2], the computationally natural subclass of *sequential* ASMs is natural also from a logical point of view, corresponding to the class of quantifier free interpretations in logic.

The tutorial and the Lipari guide incorporate the experience which had been gained through the early applications of ASMs, those described above and those reported in the contributions to the first international ASM workshop which was organized as part of the 13th World Computer Congress in Hamburg in 1994 [255], see below. The major addition of the Lipari guide to the tutorial concerns the notion of distributed ASM runs. In [190] an ASM model had been developed for the parallelism of Occam. It was presented by Gurevich in May 1990 in another series of lectures in Pisa, which inspired the concrete theme for Riccobene's PhD thesis [264] to refine the ASM model for Prolog by the different forms of parallelism encountered in Parlog, Concurrent Prolog, Guarded Horn Clauses, and Pandora [91, 92, 265]. Later another instance of refinement and parallelization of Prolog to a semi-ring based constraint system appears [32], replacing the Call and Select Rules of [44] by a Reduction Rule which activates a child process simultaneously for each alternative of the current process. The notion of parallelism in these models was generalized in [162] where ASM models appear for the Chemical Abstract Machine and the π-calculus. Eventually in 1995, the Lipari guide definition of distributed ASM runs supersedes these more restricted definitions of concurrency for ASMs.

4 Testing the Practicability of ASMs

Once the practical potential of the ASM notion had been understood, a natural next step was to test its practical impact by trying out ASMs for the modeling and a rigorous mathematical and experimental analysis of a variety of complex real-life computing devices, looking for relevant problems beyond those of the semantics of programming languages. In the Fall of 1992 Börger defined this program and started it with his students by systematically extending the application of ASMs to the specification and analysis of virtual machines, processor architectures, protocols, embedded control software and requirements capture. As part of this effort the 1993 Lipari Summer School on *Specification and Validation Methods* [51] was organized wich triggered the fundamental Lipari Guide [181] and a series of papers on applications of ASMs [102, 303, 83, 204, 189][6]. The endeavor attracted many researchers and resulted in the elaboration of a

[5] The initially present construct to shrink domains, which was motivated by concerns about resource bounds, was abandoned because it belongs to garbage collection rather than to high-level specification. Some technical variation was later introduced concerning the treatment of non-determinism and of inconsistent update sets. For a more substantial recent extension see [196].

full-fledged system design and anlysis method built upon the notion of ASMs, pragmatically confirming the ASM thesis and the choices Gurevich made for the definition of ASMs in the Lipari guide.

4.1 Architecture Design and Virtual Machines

For computer architectures, the work on the program started during the winter term 1992/93 with a reverse engineering study, commissioned by a group of physicists in Pisa and Rome for the massively parallel APE100 architecture, a rather successful dedicated machine which had been developed for floating point intensive numerical simulations in Lattice Gauge Theory [19, 20]. As part of the work for Del Castillo's Tesi di Laurea, a programmer's view ground model has been defined in [71] and refined in [70] to a provably correct decomposition of the control unit processor zCPU, a VLSI implemented microprocessor with pipelining and VLIW parallelism, built from formally specified basic architectural components. The intermediate models, obtained by stepwise refinement, correspond to views of the architecture provided by different languages used within the APE100 compilation chain, a crucial part of the software environment of the machine. A companion Tesi di Laurea [88] had the goal to isolate the underlying pipelining scheme and to prove its correctness via stepwise refined models. Hennessy and Patterson's RISC machine DLX was chosen as reference architecture to deal with the standard pipelining methods which handle structural hazards, data hazards, and control hazards respectively. Starting from the one-instruction-at-a-time view of the processor, each hazard is dealt with in one dedicated refinement step which concentrates on the corresponding machine property. This ASM based architecture verification method was taken up by other research projects which are mentioned below.

For virtual machines, the program to extend the WAM work [100] beyond issues concerning the implementation of programming languages started with modeling the well known Oak National Laboratory public domain software system PVM [151], a general purpose environment for heterogeneous distributed computing. The virtual machine appears there logically as a single distributed-memory computer, "created" by PVM out of a dynamic heterogeneous set of physically interconnected and concurrently operating machines, namely host computers which can be dynamically added to or deleted from the virtual machine and may belong to a variety of architectures (including serial, parallel, and vector computers). Glässer suggested to try out the notion of distributed ASMs for modeling PVM. In [75, 76] a ground model for the Parallel Virtual Machine is defined at the C-interface level, where it appears as a distributed ASM with a

[6] A companion Lipari Summer School on *Architecture Design and Validation Methods* followed in 1997 [61]. Another one is forthcoming in 2002 on *Software Engineering*, with ASM based courses held by Riccobene, Gurevich, Börger.

characteristic event handling mechanism and message-passing interface (reflecting the uniform access the PVM agents ("daemons") have to daemons on other host machines, whether multicast or point-to-point between single tasks).

The cooperation on modeling the PVM triggered the project to provide a ground model for the at the time new IEEE standard VHDL'93 [208] of the hardware design language VHDL. The models defined in [79, 80] come as a distributed ASM and cover the entire language with the new features of the 1993 standard, in particular the complete signal behavior and the time model, including pulse rejection limits and the various wait and signal assignment statements involved in the subtle issues related to postponed processes. Later these ASM models have been used in W. Müller's PhD thesis at the University of Paderborn [245] for defining the semantics of a pictorial extension PHDL of VHDL'93, by a group of Toshiba engineers for an extension to analog VHDL and Verilog [271, 272, 268, 269, 270], and recently for an adaptation to SystemC [247] and to SpecC [246].

Conversations with Langmaack since 1991 on the relations between the ASM based method used for proving the correctness of Prolog-to-Wam compilation [100] and the European ProCoS project on provably correct systems [230], which aimed in particular at a correctness proof for executing compiled Occam programs [248] on the Transputer architecture [170], have led to the Transputer verification ASM case study. In [73] the Transputer instruction set architecture has been modeled by a hierarchy of stepwise refined ASMs to support the correctness and completeness proof for the general compilation scheme of Occam programs to Transputer code proposed in [209, 210]. As basis for the semantics of truly concurrent and non-deterministic Occam programs an appropriate ground model ASM has been used, which was defined in [74]. It leads from the programmer level by various proven to be correct refinement steps to the starting point of the hierarchy of Transputer models, namely a machine which appears as an abstract processor running a high and a low priority queue of Occam processes. This ASM based method for a mathematical verification of real-life compilation schemes with respect to a rigorous semantics of source and target languages has been taken up again in the Verifix project discussed below, based upon the recognition in the ProCoS project that to establish the correctness of (modulo hardware correctness) reliable initial compilers, in addition to verifying the compiling function also the verification of a compiler implementation is needed.

4.2 Protocols

Using ASMs for modeling and verifying protocols has been started in three papers which were published in [51]. The work in [83] is an answer to one of the at the time frequent public challenges of ASMs: Abraham and Magidor at a

Dagstuhl seminar in June 1993 expressed doubt that the atomic character of function updates in ASM transitions would prevent these machines from naturally reflecting complex combinations of low level durative actions. The discussion in the seminar focussed on the concrete example of Lamport's mutual exclusion protocol known as bakery algorithm [228, 229] for which Abraham had presented a new proof method, relying on a distinction between a lower and a higher view of the algorithm [2, 3]. In [83] three ASMs are built. The first one serves as a ground model to faithfully reflect Lamport's protocol. By abstracting from the low-level read and write operations of the ground model, a high-level model with atomic actions (non-overlapping reads and writes) is defined and then

– proven to have the desired correctness and liveness properties (under four natural assumptions on the abstract functions that were used),
– proven to be correctly refined by the ground model (proving that the assumptions made for the abstract model hold for the implementation).

In the third ASM, the state of the abstract machine is refined by replacing atomic actions with durative ones, allowing overlapping of reads and writes to shared registers. It is proved that this refined notion of state satisfies the corresponding assumptions made for the machine with atomic actions. The resulting correctness and liveness proofs for the bakery algorithm considerably shorten numerous other proofs in the literature. The arguments are expressed using a mapping of ASM moves to moments of continuous and linear real-time, but they can and have been rephrased in the more general terms of partial orders [195] which characterize the notion of distributed ASM runs in the Lipari guide. A further analysis of the role of timing constraints on distributed ASM runs for proving the correctness of refinements of distributed asynchronous algorithms with continuous time appears in [118].

In [204] the technique of successive refinements of ASMs is applied to show how one can provide for a real-world protocol a faithful readable specification together with an understandable correctness proof. The Kermit file-transfer protocol chosen as object of the study had been presented by Knuth in his foreward to the Kermit book [120] by expressing the

> hope that many readers of this book will be challenged to find high-level concepts and invariant relations by which various versions of the Kermit protocol can be proved correct in a mathematical sense.

In fact Huggins deals with a complete version of Kermit, showing how this protocol combines the underlying alternating bit protocol and its sliding windows extensions, thus differing from numerous earlier verification studies in the literature which had focussed on these two simple protocols in isolation. Later two other ASM formalizations of the alternating bit protocol alone appear, one in [235, Ch.5] as an illustration of a modularization and communication concept implemented on top of ASMs, the other one in [173] to illustrate the application of algebraic-categorical composition schemes to ASMs.

In [189] a processor group membership protocol is modeled as distributed ASM, a typical protocol of the kind used to achieve fault tolerance for distributed computing services. The underlying assumptions for the well-functioning of the protocol are made explicit, such as the reliability of the message passing mechanism, lower bounds for processor recovery, upper bounds for message exchange time, etc. These assumptions are then proved to imply the correctness of the protocol, namely that despite of delays in message passing and server failures, the protocol achieves a global agreement about the set of all correctly functioning processors in a synchronous system.

This debut of ASMs for protocol verification triggered numerous other ASM projects in the area which are discussed below.

4.3 Why use ASMs for Hw/Sw Engineering?

In the first half of the 90'ies, the concept of ASMs encountered considerable scepticism and not seldom strong opposition in the scientific community, even in Europe. Interestingly enough the criticism came from two directions, on the one side from researchers whose longstanding trust in purely declarative logico-algebraic methods made them view ASMs as nothing else than yet another form of an old fashioned low-level operational method, on the other side from researchers who claimed earlier fathership for the notion in a variety of forms. Both objections contain a grain of truth. They motivated the attempt, made in 1995 [53] and again in 1998 [59], to better understand the relation between ASMs and established formal methods and to formulate the stringent scientific reasons why after decades of intensive research in the area, an apparently new concept is proposed as basis for a practical high level system design and analysis method. The articles explain that although a logician discovered the concept of ASMs, as an outsider driven by a foundational concern[7], the *notion* triggered the development of a *method* which allows one to really "complete the longstanding structural programming endeavour (see [121]) by lifting it from particular machine or programming notation to truly abstract programming on arbitrary structures" [59, Section 3.1]. As a matter of fact the notion is made up of two ingredients which had been there already for a long time but with no Columbus to bring them together for further exploration of the New World, namely Dijkstra's basic concept of *abstract machines* [134] and the fundamental idea to use Tarski structures as most general notion of *abstract states*, an idea which pervaded the area of abstract data types and algebraic specifications [254, 146]:

$$\text{ASM} = \textbf{A}\text{bstract } \textbf{S}\text{tate} + \text{Abstract } \textbf{M}\text{achine}.$$

5 Making ASMs fit for their Industrial Deployment

The positive experience gained through the multitude and diversity of successful ASM-based modeling and verification projects convinced Börger of the potential of ASMs for industrial system development and brought him to the decision, in the Spring of 1994, to apply ASMs to down-to-earth software engineering problems and to pave the way for their industrial deployment. At the time it was hard to find support for projects directed towards this goal, the effort was judged even by some leading peers to be a waste of time. Nevertheless it attracted more and more researchers and eventually led to an industrially viable, theoretically well-founded *system development method built around the concept of ASM*, an approach which supports practical system design and analysis by application-tailored high-level modeling that is seamlessly linked to executable code, going through mathematically verifiable, experimentally validatable, and objectively documentable refinement steps.

The program was articulated through the following three major themes surveyed below:

- investigation of *practically relevant case studies* and system development problems, to identify the strengths and weaknesses of ASMs for software development, and to compare ASMs with established formal methods,
- application of ASMs in challenging *industrial software engineering projects*,
- *integration of tools* for simulation, verification, documentation, and maintenance of ASMs during the software development process.

5.1 Practical Case Studies

In the Spring of 1994 the preparation of a research competition among methods for semantics and specification was started with the declared goal to "contribute to a realistic comparison, from the point of view of practicality for applications under industrial constraints, of the major techniques which are currently available for formally supported specification, design, and verification of large programs and complex systems" [7, pg.1]. This developed into the Steam Boiler Dagstuhl Seminar [5]—by the name of the industrial case study the participants were asked to solve—and into the Steam Boiler Case Study Book [6] which came out of a subsequent international call for participation. Although only a quarter of the numerous solutions came up with a validatable executable model, to

[7] In an e-mail of September 13, 1996, addressed to the ASM community at ea@ira.uka.de, Gurevich stresses the epistemological point that "the core of evolving algebras is an observation (or discovery if you will) and not invention. That is what the EA thesis is all about."

provide a complete ASM solution turned out to be a relatively easy task. In [29] first a ground model ASM is defined—a rigorous form of the given problem description which is phrased in such a way that it can be checked by the domain expert to be faithful to the intended requirements. Then the ground model is stepwise refined to C++ code, each intermediate model reflecting some design decision[8]. Proofs for some of the required system properties are reported, and during a demo to the seminar, Durdanovic showed his C++ program to successfully control the simulator [233] Lötzbeyer had built at FZI in Karlsruhe for an experimental evaluation of the problem solutions.

Mearelli's Tesi di Laurea, started at the beginning of 1995, had the goal to extend the positive experience made with the development of the steam boiler control software by a test of the integratability of the ASM method into the various phases of the software development cycle. As experimental guide an ASM ground model for the at the time freshly published Production Cell Case Study [231] was developed and stepwise refined to C++ code (see [89, 237]), with particular attention to the following two features:

– the modularity of the specification and the code and their structural similarity (to support code inspection), together with their complete documentation as support for inexpensive changes and easy maintenance,

– the applicability of standard verification and validation methods to prove the desired properties stated in the requirements.

In fact in [89] all the required correctness, safety, performance, and liveness conditions are proved by mathematical argument—typically for the high-level model under appropriate assumptions, proving these assumptions to hold at the refined level. The C++ code produced by implementing the final refined ASM model [237], taking care that the specification can be traced through the structure of the code, has been validated by extensive experimentation with the simulator built at the FZI in Karlsruhe. It has also been submitted for an inspection process to another software engineering Dagstuhl Seminar, organized in 1997 and focused on "Practical Methods for Code Documentation and Inspection" [86]. To test the integratability of mechanical verification methods into the software development with ASMs, the Production Cell models were used for model checking experiments [306, 256] and for theorem proving with PVS [147]. As one of the first test examples for his code generator from ASM specifications (see below), Schmid has used the Production Cell ASM for generating efficient C++ code whose structure allows one to trace the specification to support the reliability of code inspection [278].

[8] One can view it also the other way round as lifting the C++ code to a more abstract level with simultaneous updates, access to historical function values, etc., a methodological view which was held by Durdanovic and has been further elaborated in [124].

In 1999, through another software engineering Dagstuhl seminar, it has been tried to bring together once more researchers from academia and practitioners from the software industry to evaluate the contribution of so-called formal and informal methods for solving practical system engineering problems. The seminar was focused on problems encountered in industrial software development processes to capture, document, and validate requirements in a principled manner [84]. This seminar, too, was centered around a practical case study which triggered some complete solutions published in [81]. The ASM solution of this Light Control Case Study [94] showed that building and simulating ASM ground models is an efficient method to capture, validate and document requirements for a precise reason: it allows one to document in a traceable way the desired mix of rigorous, explicit ("formal") elements of description and of others intended to remain vague, implicit ("informal"). Such a mix is needed to bridge the gap between the views of the application domain expert and of the system designer, persons who speak different languages but nevertheless have to understand each other to be able to agree on the definite characteristics of the system to be developed. This observation has led to Cavarra's PhD project [116] where an attempt is made to link ASMs to so-called semi-formal specification techniques as they are used in industrial practice. The formal versus semi-formal issue is an instance of the more general need for an encompassing framework to combine heterogeneous specification elements, which is discussed in [155, 9, 125]. The 1999 Dagstuhl Seminar [84] brought out this requirements engineering aspect of the systematic *separation of different concerns* which has been advocated in [53, Sect.4] as a characteristic and major distinctive feature of the ASM method compared to other approaches to system design (see also [50, 180]). It was pointed out again in [60, Sect.1] that such a separation of orthogonal system features, and of different methods to model and analyze them, is necessary for a successful combination of multiple ways to construct and relate different system views—by modeling, simulating, and verifying the system with different degrees of precision. Indeed it is one of the reasons for the success of the ASM method that the mathematical "openness" of the basic ASM concept allows fine-tuning this *separation-for-integration strategy*—a classical divide-et-impera approach—to the needs of the system to be developed or investigated.

Also other case studies which had been presented during this period as challenges to the scientific community have been elaborated using ASMs. See the ASM solution [205] to the Broy-Lamport specification problem [113] which had been formulated in 1994 for the Dagstuhl seminar on reactive systems. Another example is the real-time based ASM modeling and verification in [186, 21, 22] of the Railroad Crossing Problem to which Heitmeyer and Mandrioli's book [202] was dedicated.

5.2 Industrial Pilot Projects and Further Applications

ASMs at Siemens/Munich. For his sabbatical year 1995/96 Börger chose the goal to find out whether the above described applications of ASMs to requirements capture and to design and analysis of control software scale to the needs of industrial design environments. This developed into a fruitful cooperation during the years 1996-1999 with Päppinghaus at Siemens in Munich, largely focussed on design methods for railway-related software [65]. It led to a rather successful application of ASMs in a middle-sized industrial software development project (FALKO, May 1998–March 1999, reported in [90]). The salient methodological outcome of this cooperation was the creation of a *prototypical ASM based industrial development environment* which supports a seamless flow from the definition of an ASM ground model to compilable (in the specific case C++) code and its maintenance.

Obviously, to make this project succeed, appropriate ASM tools had to be created and used extensively. In the Spring of 1995 Del Castillo had started his PhD work, located at the university of Paderborn, to build a tool environment for the specification and simulation of ASMs. The FALKO ground model was formulated in the ASM-SL language Del Castillo meanwhile had defined for the ASM Workbench [123], so that in the FALKO design phase early versions of this machine could be used for extensive testing of the high-level FALKO models, prior to coding. At the end of the design phase, as part of his PhD project started in the summer of 1998 at Siemens Corporate Research in Munich, Schmid developed a compiler from ASM-SL to C++ [278]—it generated the program which since March 1999 is in daily, failure-free use by the Vienna Subway Operator for the validation of subway operational services. For documentation and maintenance purposes Schmid developed a literate programming tool allowing to keep a single collection of consistent HTML documents from which the ASM-SL code can be extracted as input to the ASM Workbench or to the compiler, but also in pretty-printed form for the human reader.

In the third part of his PhD work [279, Ch.2], Schmid successfully applied this tool-supported structured ASM modeling and refinement technique also in a large ASIC design and verification project at Siemens München. This includes the definition of a notion of ASM components which was used for the behavioral specification of digital hardware circuits, and of the development of a compiler from ASM components to VHDL.

CO-Monitoring System. Another early industrial application of tool supported ASMs, namely for an automated fire detection system adopted by three major German coal mines, is reported in [114]. Kappel's Prolog based interpreter for sequential ASMs [219] has been extended for this project to support the parallel execution of independent modules, representing distributed processes which are synchronized via stream based communication. The extension comes with a visualization mechanism for run data.

DFG Projects Deduktion/Verifix. In 1994 Börger suggested to the German Research Council project "Deduktion" to apply mechanical proof verification for proving properties of ASM models of real-life programs [280]. In particular, some of the refinement steps in the above mentioned WAM correctness proof have been mechanically verified using Isabelle [262], whereas using KIV the entire proof has been elaborated for a mechanical check, using not only all the 12 refinement steps from [100], but adding one more intermediate model to make the proof feasible for the machine [276, 274]. In [275] a scheme is extracted from that work for proving the correctness of ASM refinements using generalized forward simulation. This use of ASMs for proving the correctness of compilation schemes has been further developed in a part of the Verifix project [164] of the German Research Council where instead of schemes for compilation into virtual machine code the correctness of concrete compilers compiling into real-life machine languages is investigated. To mention only a few examples from the subpart of the Verifix project where ASMs are used, which appeared in [149, 318, 136, 150, 200, 201, 167, 199]: a ground model ASM for the DEC-Alpha processor family has been extracted from the manufacturer's handbook; compiler back-ends have been built based on realistic intermediate languages to prove their correctness, using generic PVS theories developed in [135] to define refinement relations between ASMs; ASMs have been used to describe compilers which verify the correctness of the code they generate, etc. For the specification of source languages also Montages (see below) has been used, adopting however attribute grammars to formulate static semantics features. In [238] appropriate ASM models are defined to prove the correctness for the static link technique.

Montages at ETH Zürich. A related research effort has been undertaken at ETH Zürich, triggered by Gurevich's ASM-lectures delivered there in the Spring of 1995. It was driven by the Montages project [224, 11], geared to support, by an appropriate combination of graphical and textual elements, the simultaneous specification of the static and dynamic semantics of programming languages, exploiting the syntax-driven modularity which is typical for sequential languages where instructions are executed one after the other and one per step. The method is illustrated by a complete definition of syntax, static and dynamic semantics of Oberon in [225] and of C in [207]. In [10] a development tool (Gem-Mex) for creating Montages is presented which has been applied to provide an executable semantics for Mosses' Action Notation [9]. A successful application of Montages to the design and specification of a domain-specific language for a Swiss bank is reported in [226].

SDL-2000 Standard. Another remarkable industrial exploitation of ASMs comes with the abstract operational ASM definition of the new industrial standard for the design language SDL, widely used for over 20 years to develop distributed systems. In November 2000, the international standardization body ITU-T for telecommunications accepted the ASM model as official definition of the 2000 standard of the language. The project of modeling the intricate static and dynamic semantics of the distributed real-time features of SDL as executable ASMs, in the context of rich data and hierarchical control structures together with advanced object-oriented and exception handling features, was started in [160], proposed to the SDL Forum in [156], and led through three years of intensive work of a body of experts [157, 158, 140, 261] to the complete standard definition [212]. It covers the static and the dynamic part of the semantics, currently an SDL-to-ASM compiler and further tool support of the ASM model for the standard are in development [141].

ASM Analysis of Java and the JVM. The project to apply ASMs to a systematic investigation of Java and its implementation on the Java Virtual Machine was born from a debate, in March 1997 in Dagstuhl, on *How to use Abstract State Machines in Software Engineering* [56]. The modeling experiments during the first two years [106, 107, 108, 109, 110] were geared to establish through this concrete real-life case study the respective merits of functional, axiomatic, and operational abstract state-based specification features. They were followed by two more years of mathematical and experimental analysis, streamlining and structuring the ASM models of Java and of the Java Virtual Machine, and adding correctness and completeness proofs for a standard compiler of Java programs to JVM code and for the security critical bytecode verifier component of the JVM [291]. The technique of structuring the ASM models into language layers and machine components [62] is based upon the composition concepts which were developed in [103]. It led to a natural refinement of the high-level Java/JVM models to AsmGofer executable models [277] which can be used for code testing purposes. In a recent evaluation of about 40 Java/JVM research projects worldwide it is stated that “the Jbook (i.e. [291]) ... gives the most comprehensive and consistent formal account of the combination of Java and the JVM, to date”[198].

Architecture Projects. Mention has been made already above of the industrial extensions of the ASM models for VHDL [79, 80] to analog VHDL and Verilog [271, 272, 268, 269, 270], to PHDL [245], to SystemC [247] and to SpecC [246], as well as of Schmid’s compiler from ASM components to VHDL [279, Ch.2]. The ASM modeling method for instruction set architectures developed in the APE100 project [70] has been enhanced in [126, 127] to instrument models to collect data for evaluating design alternatives.

In [153, 292] some steps were taken to mechanically verify the pipelining correctness proof using the KIV system and PVS, but unfortunately without covering the complete hierarchy of four models in [88], so that an omission of a hazard case in the last refinement step remained undetected until Hinrichsen found it during his work on generating pipelined systems from sequential processor specifications [203]. The design and verification method of [88] has been applied in [206] to an advanced commercial RISC processor with a simpler pipelining scheme. It is reused in [297] to illustrate an approach to automatically transform register transfer descriptions of microprocessors into XASM-executable ASMs [8], thus allowing to generate a simulator for a processor architecture from its netlist description or from a graphical description of its data-path. In the same spirit, in [298] an ASM model is developed for a VLIW digital signal processor of the Texas Instruments TMS3200 C6200 family. These papers are part of an architecture and compiler co-generation project, led by Teich at the University of Paderborn, where ASMs and their execution in XASM [8] are systematically applied for hierarchical modeling of application specific instruction set processors [296].

Further ASM Applications. Since 1994 numerous advanced applications of ASMs appeared for protocol verification and in other fields of computer science, namely formal grammars, databases, electronic commerce, software architecture, finite model theory, complexity theory. In [217, 241, 242] distributed ASMs are used to model various formal and natural language grammars. In [193] the database undo-redo recovery algorithm is modeled at several levels of abstraction, showing the ground model to be correct and proving the correctness of each of the four refinement steps, leading to a model incorporating cache and log management. In [24] ASMs are used to describe the semantics of a domain-specific language, tailored to program the control for event driven database applications. In [144] an ASM based prototype system is described for specifying electronic commerce applications. The contribution of ASMs here is to provide a rigorous transparent way for describing the state changes involved in electronic commerce negotiations, concerning the traded products, the negotiators, their orders, the laws accepted as basis for the particular negotiation, etc. This paper and its companion paper [1] triggered the recent study of decision problems for restricted classes of relational ASM-transducers [288]. In [218] ASMs are used to specify a name management model. In [14] ASMs are used to define the semantics of patterns and for correctness proofs for workarounds. In [148, 17, 171, 152] ASMs are exploited for dealing with testing issues, taking up a suggestion made in [59, page 36] and [60, page 6]. In [295] an interesting ASM-based approach to software architecture design is proposed, allowing to specify software systems by appropriately connecting components which are characterized abstractly in terms of the services they export or import. In [282, 235] interacting ASMs are defined. In [117] the task and data parallelism of the programming language P3L is analysed on the basis of an ASM model. In [250] the theme of modeling PVM [75, 76] is taken up again using ASMs to propose a definition for the semantics of grid systems.

The early ASM specifications and verifications of protocols have been continued in [30] where the Kerberos Authentication System is modeld by a hierarchy of proven to be correct stepwise refined ASMs, disclosing the minimum assumptions to guarantee the correctness of the system as well as its security weaknesses. In [31] the Needham-Schroeder protocol is analyzed to illustrate a general ASM framework for a practical analysis of cryptographic protocols. Stroetmann defined a ground model ASM for the constrained shortest path problem and proved it to be correct from a small number of natural axioms [294]. He then refined the ground model in a proven to be correct way down to (but excluding) the level of an efficient proprietary Siemens implementation of the algorithm in C++. During his research stay in Pisa in 1997/98, Durand investigated the cache coherence protocol in the Stanford FLASH multiprocessor system. In [137] a high-level ASM specification and a correctness proof are provided which detected some incoherent and incomplete features in the given protocol description. This model has been used in [307] as case study for a real-life application of model checking to ASMs, using the SMV model checker. In [139] a two-level ASM specification of a distributed termination detection algorithm is given together with an equivalence proof between the two machines. In [187] it is illustrated by two ASMs for an algorithm proposed by Lamport that the notion of equivalence of algorithms depends on the level of abstraction at which the algorithm is viewed.

Foundational Progress. It became characteristic for ASM workshops and collections of ASM papers [255, 51, 57, 58, 161, 188, 78, 77, 35] to contain both theoretical and practical contributions. In fact in addition to the theoretical papers mentioned already above, there have been important contributions of ASMs also in complexity and finite model theory, although the surprising observation expressed in [319] is still true, namely that the theoretical potential of ASMs has been recognized and explored to a less extent than their practical applications. In [36] ASMs are used as computation model which can reflect the practical experience that for real-life computations, constant factors matter. Based upon this model, linear-time hierarchy theorems for random access machines and ASMs are proven. In particular it is shown that there exists a sequential ASM U and a constant c such that, under honest time counting, U simulates every other sequential ASM in lock-step with log factor c.

In [168] finite model theory is extended to metafinite models, covering the mixture of finite and potentially infinite features as they appear typically in practical applications in the states of an ASM. [38] is an investigation into the notion of the reserve set of an (untyped) ASM, exploring the ideas of adding structure within the reserve and the non-determinism of importing new elements.

In [41, 39, 42] a polynomial time version of ASMs is defined and investigated which captures the portion of the complexity class PTIME where parallel algorithms (with arbitrary finite structures as inputs) are not allowed arbitrary choice. In [43] this choiceless polynomial-time variant of ASMs is explored as a query language for relational databases. In [169, 287] a restriction is defined to capture log-space computable functions on structures. The ASM choice construct *(choose* $x : F(x)$*)* motivated also [37] where extensions of first-order logic with the choice construct are studied.

[286, 289] deal with decision problems for restricted classes of ASMs.

Recent industrial ASM applications. Since the Fall of 1998 when Gurevich joined Microsoft Research, various applications of ASMs within Microsoft have been reported. The first one [197] is an ASM specification of the Windows Card Runtime Environment with a verification of certain safety properties. During 1999/2000, a command-line debugger of a stack-based runtime environment has been reverse engineered from the given 30k lines of C++ code, using three successive abstraction steps: one to extract the ground model which defines the same core functionality as the debugger, one to reflect the compile time structure of the underlying architecture (of modules of classes containing functions containing code) together with a restricted view of the run time structure, and eventually a control state ASM focussing on the interaction between the command issuing user and the reacting runtime environment. The conclusion reads [15, pg.367]: "The study provides evidence for ASMs being a suitable tool for building executable models of software systems on various abstraction levels, with precise refinement relationships connecting the models." In [159] a distributed real-time constrained ASM has been developed to specify the Universal Plug and Play (UPnP) architecture for peer-to-peer network connectivity of intelligent devices. A refined model is derived which is executable in AsmL (see below) and can be used to inspect ASM runs at the required level of detail for conformance testing.

In [308] an ongoing industrial project in Australia is mentioned where ASMs are used to specify and model check a railway interlocking system.

5.3 Tool Integration

Already at the times of the very first industrial application of ASMs in the context of the ISO standardization of Prolog, the issue of how to turn the abstract specifications into executable ones for high-level simulations prior to coding has been recognized as crucial. Since then it has been resolved in various ways, as we are going to survey in this section. The progress made over the years in using ASMs in industrial projects for building models of software systems on various abstraction levels made it clear however that it is not enough to build simulators. They must also be linked to standard verification, testing, refinement, versioning and maintenance methods and tools as used in current development environments.

ASM Interpreters. The first interpreter for sequential ASMs was developed in 1989/90, in Kappel's Diplom thesis at the University of Dortmund [219]. During the same time at Quintus a special interpreter for Prolog ASMs was built [112] to execute the ASM models proposed in 1990 and accepted in 1995 for the ISO standardization [69]. In 1995 an elegant 9-line Prolog interpreter for sequential ASMs appeared [23]. The same year in Oslo [131] a functional ASM interpreter is implemented. As Huggins reports (see [60, footnote 3]), at the University of Michigan an interpreter for sequential ASMs was developed (in C) by Harrison and Huggins from 1991-1994. In this context Huggins also implemented an automated partial evaluator for sequential ASMs [185] which was extended in [133]. From 1994-1996 the Michigan interpreter was upgraded by Mani to distributed ASMs, and in 1996 Gurevich invites the ASM community to "Please use the new Michigan interpreter and tell us how do you like it and what features you would like to have." [9]

In 1995, Glässer, Del Castillo and Durdanovic propose an abstract ASM machine (at the time called EAM for abstract evolving algebra machine). It is defined by stepwise refinement as a platform for implementing ASM tools and led to the design of a virtual machine architecture as a basis for a sequential implementation of the EAM [124]. This was the start for two PhD projects, both located at the University of Paderborn and with the goal to develop a practically useful tool support for ASMs. The first result to come out was Del Castillo's ASM Workbench [122, 123] which has been used in a methodologically interesting way in the FALKO project at Siemens [90]. The high-level ASMs of the to be developed railway process software were tested for some scenarios provided by the customers by running the scenarios on the Workbench, where upon calling an abstract function for some argument, the requested value is taken from its instance in the particular scenario (read from a file which describes the given use case). [138] continues the ideas developed in [124] by building an ASM Virtual Architecture as basis for a comprehensive ASM tool environment which comes up to industrial efficiency standards.

In [260] the MAX tool supporting the generation of language-specific software from formal specifications is presented which uses functional algebraic attribution techniques for the static semantics and ASMs for the dynamic semantics. The development tool Gem-Mex presented in [10] for creating Montages has been extended by Anlauff to the system XASM [8] for producing efficiently executable and easily reusable ASMs and coming with an XASM-compiler, a runtime system and a graphical debugging and animation interface. It contains a mechanism for structuring ASMs based on components which can be compiled separately and thus be put into a library for later reuse. Technically this is achieved by enhancing a static module concept, where submachines can be used as ASM

[9] e-mail to ea@ira.uka.de of September 13, 1996.

rules or as external functions, by access/update constructs which provide information on permissions to read/write locations of submachines. An application of this component concept is presented in [12]. (For a different set of modularity concepts, geared to define and refine I/O-behavior of programs according to their abstract syntax, see [165].) XASM offers an interface to C allowing a) C-functions to be used as static or monitored ASM-functions, and b) ASMs to be called from within C-programs. In [223] one can find a denotational semantics of XASM. The applications of XASM for Montages and Teich's architecture and compiler co-generation project are described above.

The development of AsmGofer [277] was driven by the goal to provide for executability of the models for Java and the JVM defined in [291], although it represents a general interpreter for a large class of structured ASMs (see e.g. its use in the Light Control case study [94], or the use of its variant AsmHugs in [15]). It comes with GUI support, debugging facilities, and support for a literate programming technique. The structuring and composition concepts implemented in AsmGofer have been defined in [103] and have been used to decompose the Java/JVM models into language layered and functional components [62]. The definition of these concepts was driven by the double concern a) to distill some forms of standard programming constructs which are really needed in large applications, and b) to integrate them as natural standard refinements into the ASM world with simultaneous multiple updates of a global state. As a consequence these concepts are special cases of the more general algebraic concepts investigated in [236].

The MOSES tool suite in [213] is tailored for the specification and prototypical implementation of visual notations for discrete-event systems. It comes with a graph editor (from visual notation), a simulator with animator, a debugger and some management tools.

Recently a new specification language AsmL developed by the Foundations of Software Engineering group headed by Gurevich at Microsoft Research has been made accessible [191, 16]. In [16, 17, 18] some applications of AsmL within Microsoft are reported. An important new feature of AsmL is the way it exploits the abstraction potential of ASMs to offer object-oriented structuring principles. The naturalness with which object oriented features can be described by ASMs had been observed and used already in [166, 33, 243, 244, 260, 225, 214, 215] and has led to Zamulin's systematic investigation of objects and generic types for ASMs [311, 312, 313, 314, 315, 316, 317]. A related effort was made in [67, 66, 216] and in Cavarra's PhD thesis [116] to exploit ASMs for a rigorous support of object-oriented UML techniques and concepts, triggered in the summer of 1999 by an evaluation of the high-level design characteristics of UML and of ASMs in an industrial software development environment [68]. This has inspired also the work in [251] where the ASM Workbench [123] is used to define an executable semantics for UML which covers real-time aspects.

Linking ASMs to Verification Tools. The successful link of ASMs to theorem proving systems like Isabelle, KIV, PVS, and to model checking [309] has been mentioned above. An interesting variation of model-checking, applied to the railroad crossing ASM of [186], appears in [21, 22]. Recently [293] KIV has been used also for checking the programs in Sun's Java manual against their ASM specification in [106]. A description of the tableau proof method in terms of ASMs at various levels of refinement leading from the textbook level to an implementation appears in [105].

Numerous logics have been developed to formalize ASMs and to support mechanical reasoning for them. See [172, 257, 259, 281, 299, 22]. In [263] the co-induction proof scheme is justified for ASMs, characterizing them as a class of Di-algebras and proposing "the *Di-algebra* thesis which improves the Turing thesis and which corresponds directly to the evolving algebra thesis: *Real world computable algorithms coincide with algorithms specifiable by completely constraint Di-algebra specifications*" [263, Section 3]. In [290] a logic for ASMs appears which unifies some of these logics and is based on an atomic predicate for function updates and on a definedness predicate for the termination of the evaluation of ASM rules.

6 Conclusion and Lessons for the Future

The notion of ASMs provides the basis for a taxonomy of discrete systems into classes of sequential and distributed systems [63]. The epistemological appropriateness of this classification of models of computation is guaranteed by the ASM thesis and strengthened by the arguments which prove the sequential ASM thesis [183] and its extension to parallel synchronous algorithms [40] from fundamental postulates. The universality of the ASM model of computation is also pragmatically confirmed by

- the *naturalness* with which other *models of computation* can be defined as ASM instances, directly, without any extraneous encoding (but typically not vice versa) [64],
- the *flexibility* with which ASMs could be adapted to the diversity *of modeling* problems and techniques in different application *domains* and at different design *levels*,
- the generality with which ASMs support a truly *codeless form of programming*, yielding programs—in fact semantically well-defined pseudo-code—which can be ported between languages and platforms in a semantically transparent way.

Through the collaborative effort described above a rigorous and practical *high-level design and analysis method* has been elaborated, which is mathematically underpinned by the concept of ASMs and has been successfully applied for industrial system developments. The approach offers a concrete way to turn the construction and investigation of sw/hw systems into an intellectually rewarding engineering task—a noble task of applied, experimentally backed up mathematics, or of scientifically well founded engineering of computing devices if one prefers to look at it the other way round[10]. Abrial's in various respects similar B method also shares this insight that

> the task of programming (in the small as well as in the large) can be accomplished *by returning to mathematics* [4, page xi].

There are some risks the ASM method is facing, and numerous opportunities it can realize in the immediate future. First of all we have to continue to perform our work in the computer *science core*, solving real problems and not redoing ("in ASM terms") what has already been solved satisfactorily using other concepts or techniques. The ASM community must fight the tendency to isolate itself, the battlefield is computer science, not ASMs. On the other side, if we do not want the approach to fall back to a merely academic exercise, we are well adviced to capitalize the driving force computer *technology* had for the past development of ASMs. That is to say we should continue putting ASMs to use in cutting edge industrial *applications* where the interesting problems to be solved show up.[11] Our efforts in this direction have to be intensified, to permeate the research in those places where challenging hw/sw engineering problems are solved. If we want to attract talented young researchers, we have to demonstrate them that we do believe in the intellectual and practical value of unfolding the mathematical structure of complex computing devices[12]. This activity must be recognized and professed—first of all by ourselves—as not less rewarding, scientifically, than activities which are focussed on traditional mathematical themes or on tools with ever more fascinating features.

---

[10] See the following remark by an observer of ASMs, N. Shankar (e-mail of February 12, 2002 to Börger): "The ASM model imposes a logical structure on transition systems that has inspired very capable mathematicians to construct remarkably elegant proofs. It is one of the few formal notations that appeals to both mathematicians and engineers."

[11] Also the name change to *Abstract State Machines* was triggered by the concern to better render the *practical* relevance of the notion and in fact was pushed by our colleagues in industry [319]. The new name was found by Päppinghaus after two extensive community wide electronic discussions and replaced the more theoretical sounding name *evolving algebras* which itself already was a replacement of the original *dynamic structures* and later *dynamic algebras*.

[12] See the observation by Blass [34, Section 4] that clearly there is no value in spelling out fully formalized say axiomatic set theory versions of non-trivial mathematical

The ASM notion of *distributed computation* has to be elaborated and exploited more, both theoretically and practically, to provide transparent real-life patterns for communication and synchronization of multi-agent ASMs. It has to be made clear what advantage it presents over the pragmatic interleaving view of distributed computation. The stepwise *refinement method* has to be further developed and badly needs to be supported by tools, not only for simulation, but also for verification (relating proofs with different degrees of detail), testing (relating tests at different system levels), corrective maintenance and evolutionary system adaptation. The *codeless form of programming* ASMs offer has to be exploited for producing transparent software which is not only reliable—trustworthy and secure—, but can be certified (proven and checked) to have this property, all the way from its specification as a ground model to the implementation. The platform independence of abstract ASM code should be exploited for dealing with the challenges of intelligent agents, mobile code, middleware, system architectures built from components which offer and use services with heterogeneous access, typically by mobile users, and of context dependent service quality. We must aim for our tools to not replace, but *enhance established tool environments* so that they can meet advanced needs to capture and manipulate industrial design knowledge in a rigorous, electronically documented and reusable way. We should not rest until the use of ASMs has become current practice of *professional* system development and maintenance, scientifically secured to be worth the attribute *professional*.

Acknowledgement

This survey of the ASM research at the end of the last century grew out of an attempt to answer frequently asked questions of newcomers about the reasons for this or that phenomenon they observe at their first encounter with ASMs. It is the result of a systematic study of the ASM literature, of my notes and of my ASM correspondence files which had accumulated over the years. In the retrospective it turns out that during the last 15 years we went a thorny way, full of external and internal obstacles, but we succeeded to make ASMs into what they are today. My gratitude goes to the students and colleagues—they are named in the text above and in the references—who through their work created this beautiful and promising ASM based system design and analysis method. Their trust in the endeavor and their enthousiasm reward for the negative side experiences. I also thank Schloss Dagstuhl for the effective support of the three seminars in 1995, 1997, 1999 [5, 86, 84]—they helped considerably to work out

proofs, whereas explicitly writing out ASMs for complex algorithms may be worthwhile from both the mathematical and the practical point of view, namely as a vehicle to provide a correct understanding and means of error detection and correction.

how ASMs can contribute to professional software engineering practice—, for the invitations to many other seminars in the 90'ies where ASMs became the object of very helpful, often heated discussion, and last but not least for the forthcoming ASM 2002 seminar [35] which aims at bringing together ASM researchers from academia and ASM practitioners from industry. I also thank the following colleagues for information and criticism which enabled me to complete and improve earlier versions of this bibliographical account: two referees and M. Anlauff, C. Beierle, A. Dold, I. Durdanovic, R. Gotzhein, J. Huggins, J. Janneck, A. Kappel, H. Langmaack, P. Päppinghaus, E. Riccobene, G. Schellhorn, J. Schmid, N. Shankar, A. Slissenko, A. Sünbül, J. Teich, L. Thiele, K. Winter, W. Zimmermann.

References

1. S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational Transducers for Electronic Commerce. In *Proceedings of 17th ACM Symposium on Principles of Database Systems (PODS 1998)*. ACM Press, 1998.

   Relational transducers mapping sequences of input relations to sequences of output relations are proposed for high-level declarative specifications of business models. See [288] for a related class of ASM-transducers.
2. U. Abraham. *Models for Concurrency*. Gordon and Breach, 1999.
3. U. Abraham. Bakery Algorithms. Manuscript of 41 pages from University of Beer Sheva, Nov 19, 2001.
4. J-R. Abrial. *The B-Book*. Cambridge University Press, 1996.
5. J-R. Abrial, E. Börger, and H. Langmaack. *Methods for Semantics and Specification*, volume 117. Schloss Dagstuhl (Seminar No. 9523), May 1995.
6. J-R. Abrial, E. Börger, and H. Langmaack. *Formal Methods for Industrial Applications. Specifying and Programming the Steam Boiler Control*, volume 1165 of LNCS. Springer, 1996.

   Contains the problem description for the steam boiler control competition and 22 proposed solutions obtained using the major known formal methods, with text and complete documentation on the accompanying CD.
7. J-R. Abrial, E. Börger, and H. Langmaack. The Steam Boiler Case Study: Competition of Formal Program Specification and Development Methods. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications. Specifying and Programming the Steam-Boiler Control*, volume 1165 of *LNCS*, pages 1–12. Springer, 1996.

   Motivation of the steam-boiler control competition and short characterization of the 22 problem solutions appearing in the book.
8. M. Anlauff. XASM – An Extensible, Component-Based Abstract State Machines Language. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 69–90. Springer-Verlag, 2000.

   The XASM (Extensible ASM) language for producing executable ASMs is presented. A development environment for XASM systems is described. (XASM was formerly known as Aslan.) Also appears in TIK-Report 87, ETH Zürich, March 2000, 1–21.

9. M. Anlauff, S. Chakraborty, P.W. Kutter, A. Pierantonio, and L. Thiele. Generating an Action Notation environment from montages descriptions. *Software Tools and Technology Transfer, Springer*, 3:431–455, 2001.

Montages [11] are used to provide executable semantics for Action Notation. A preliminary version was published by M. Anlauff, P. Kutter, A. Pierantonio and L. Thiele under the title "Generating an Action Notation Environment from Montages Descriptions" in the Proceedings of the 2nd International Workshop on Action Semantics (AS'99), edited by P. Mosses and D. Watt, University of Aarhus, Department of Computer Science, BRICS Notes Series NS-99-3, March 1999, pages 1–42.

10. M. Anlauff, P. Kutter, and A. Pierantonio. Montages/Gem-Mex: a Meta Visual Programming Generator. TIK-Report 35, ETH Zürich, 1998.

An introduction to Montages [224] and Gem-Mex, the development tool for creating Montages (using a graphical editor) and generating language interpreters from them, supporting also debugging and animation features. A description of their use and some small examples can be found in *Formal Aspects of and Development Environments for Montages* by the same authors, published in M. Sellink (Ed.): 2nd International Workshop on the Theory and Practice of Algebraic Specifications, Springer Workshops in Computing 1997. Another description is in *Tool Support for Language Design and Prototyping with Montages* published by the same authors in Proceedings of Compiler Construction (CC'99), Springer LNCS 1999. Another illustration is in [12].

11. M. Anlauff, P. Kutter, and A. Pierantonio. Enhanced Control Flow Graphs in Montages. In D. Bjoerner and M. Broy, editors, *Proceedings of Perspectives of System Informatics (PSI'99)*, Lecture Notes in Computer Science. Springer, 1999.

A refined definition of Montages, based on the notion of finite state machines, triggered by the use of Montages for defining the static semantics of Java in [304] which showed some shortcomings of the original formulation in [224].

12. M. Anlauff, P.W. Kutter, A. Pierantonio, and Asuman Sünbül. Using domain-specific languages for the realization of component composition. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering (FASE 2000)*, volume 1783 of *Lecture Notes in Computer Science*, pages 112 – 126, 2000.

An illustration of how to apply Montages [224, 11] and of its tool environment Gem-Mex [10] for the implementation of component interaction.

13. 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 [100], 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.

14. U. Assmann, A. Heberle, W. Löwe, A. Ludwig, and R. Neumann. Language Concepts and Desing Patterns. Manuscript, 1999.

ASMs are used to define the semantics of patterns and for correctness proofs for workarounds.

15. M. Barnett, E. Börger, Y. Gurevich, W. Schulte, and M. Veanes. Using Abstract State Machines at Microsoft: A Case Study. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 367–380. Springer-Verlag, 2000.

A description of a reverse engineering case study, modeling a command-line debugger of a stack-based runtime environment at three levels of abstraction.

16. M. Barnett and C. Campbell and W. Schulte and M. Veanes. Specification, Simulation and Testing of COM Components using Abstract State Machines. In R. Moreno-Díaz and A. Quesada-Arencibia, editors, *Formal Methods and Tools for Computer Science (Local Proceedings of Eurocast 2001)*, pages 266–270, Canary Islands, Spain, February 2001. Universidad de Las Palmas de Gran Canaria.

A description of the use of AsmL to specify, simulate, and test the interfaces of Microsoft COM components.

17. M. Barnett, L. Nachmanson, and W. Schulte. Conformance Checking of Components Against Their Non-deterministic Specifications. Technical Report MSR-TR-2001-56, Microsoft Research, June 2001.

A method for testing a Microsoft COM (Component Object Model) component against a (possibly non-deterministic) ASM specification is presented. See [171].

18. M. Barnett and W. Schulte. The ABCs of Specification: AsmL, Behavior, and Components. *Informatica*, 17, 2002.

AsmL is shown to provide behavioral interfaces for components which allow to understand the meaning of an implementation without accessing the source code.

19. A. Bartoloni et al. A Hardware Implementation of the APE100 Architecture. *International J. of Modern Phsysics*, C(4):969, 1993.

The architecture which has been chosen by Börger for Del Castillo's Tesi di Laurea to try out ASMs for modeling real-world architectures, in the context of reengineering project for this machine which had been launched by a group of physicists in Pisa and Rome, see [71],[70].

20. A. Bartoloni et al. The Software of the APE100 Architecture. *International J. of Modern Phsysics*, C(4):955, 1993.

see comment to [19].

21. D. Beauquier and A. Slissenko. The Railroad Crossing Problem: Towards Semantics of Timed Algorithms and their Model-Checking in High-Level Languages. In M. Bidoit and M. Dauchet, editors, *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE*, volume 1214 of *LNCS*, pages 201–212. Springer, 1997.

A semantics of ASMs with continuous time using infinitesimals is defined, their runs are decribed in some first order logic. The framework is used to discuss the verification of the railroad crossing problem, based upon its ASM specification in [186]. An early version appeared in 1996 as Technical Report 96-10 of Dept. of Informatics, Université Paris 12. For a continuation see [22].

22. D. Beauquier and A. Slissenko. A first order logic for specification of timed algorithms: basic properties and a decidable class. *Annals of Pure and Applied Logic*, 113(1-3):13–52, 2001.

A continuation of [21]. The authors define a) a class of algorithms (a modified version of ASMs) with explicit continuous time, and b) a First Order Timed Logic which suffices to write requirements specifications close to natural language, enhancing the logic in [21]. The timed logic description of the semantics of that class of ASMs can be viewed as a basic set of inductive invariants for proving properties of timed ASMs. The authors consider a decidable class of verification problems and outline a compact verification proof of the Generalized Railroad Crossing Problem [186]. A first version of this work appeared under the title *On Semantics of Algorithms with Continuous Time* in October 1997 as TR 97-15 of

Dept. of Informatics, Université Paris 12. A survey of that TR and of [21] appears under the title *Verification of Timed Algorithms: Gurevich Abstract State Machines versus First Order Timed Logic* in Y. Gurevich and P. Kutter and M. Odersky and L. Thiele(Eds): Abstract State Machines – ASM 2000, International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings, March 2000, ETH Zürich TIK-Report 87, pages 22–39. Continuation in TR-00-23 of June 2000, Université Paris-12, by the same authors and with the title *A First Order Logic for Specification of Timed Algorithms: Basic Properties and a Deciadable Class*. For a set of more efficient inductive invariants for ASMs and a short formal proof of the Generalized Railroad Crossing Problem along the same lines see Technical Report TR–00–25, Université Paris 12, Department of Informatics, 2000, available at http://www.univ-paris12.fr/lacl/.

23. 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.

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

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

25. 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.

    Develops a general implementation scheme for CLP(X) over an unspecified constraint domain X, namely by designing a generic extension WAM(X) of the Warren Abstract Machine and a corresponding generic compilation scheme of CLP(X) programs to WAM(X) code. The scheme is based on the specification and correctness proof for compilation of Prolog programs in [100] and on joint work with Börger, see [26, 27, 28].

26. 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 specification and correctness proof for compiling Prolog to WAM [100] 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. [28] 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 [27].

27. 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 [26].

28. 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 [27] 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.

29. 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 use of ASMs for reliable software development see [89, 94].

30. G. Bella and E. Riccobene. Formal Analysis of the Kerberos Authentication System. *Journal of Universal Computer Science*, 3(12):1337–1381, 1997.

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

31. G. Bella and E. Riccobene. A Realistic Environment for Crypto-Protocol Analyses by ASMs. In U. Glässer and P. Schmitt, editor, *Proceedings of the Fifth International Workshop on Abstract State Machines*, pages 127–138. Magdeburg University, 1998.

ASMs are used to give a model of a general, realistic environment in which cryptographic protocols can be faithfully analyzed. The Needham-Schroeder protocol is investigated as an example.

32. S. Bistarelli and E. Riccobene. An Operational Model for the SCLP Language. ILPS Workshop on Tools and Environments for CLP held in Port Jefferson USA, 1997.

Refinement and parallelization of the ASM model for Prolog to a semi-ring based constraint system, replacing the Call and Select rules of [44] by a Reduction rule which activates a child process simultaneously for each alternative of the current process.

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

A reduced version of Smalltalk is formalized by sequential ASMs. A Hoare-style proof system is defined for reasoning about storage allocation and deallocation in ASMs. Missing constructs concern processes, inheritance, memory allocation and deallocation. Thesis supervised by Gurevich.

34. A. Blass. Abstract State Machines and Pure Mathematics. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 9–21. Springer-Verlag, 2000.

A discussion of connections, similarities, and differences between concepts and issues arising in the study of ASMs and those of set theory and logic.

35. A. Blass, E. Börger, and Y. Gurevich. *Abstract State Machines* , volume NN for Seminar 02101. Schloss Dagstuhl, March 2002.
36. 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 depends on the computation model. In 1992, N. Jones designed some computation models where the linear-speed-up theorem fails and linear-time computable functions form a proper hierarchy. The linear time of these models is restrictive. In this paper 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. One long-term goal of this line of research is to prove linear lower bounds for linear time problems. The result has been anounced 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.

37. A. Blass and Y. Gurevich. The Logic of Choice. *Journal of Symbolic Logic*, 65(3):1264–1310, September 2000.

Motivated by the choice construct of ASMs, extensions of first-order logic with the choice construct *(choose* $x : F(x)$*)* are studied. Some results about Hilbert's ϵ operator are proven. The main part of the paper concerns the case where all choices are independent. Previously appeared as Technical Report CSE-TR-369-98, EECS Dept., University of Michigan, 1998.

38. A. Blass and Y. Gurevich. Background, Reserve, and Gandy Machines. In P. Clote and H. Schwichtenberg, editors, *Computer Science Logic (Proceedings of CSL 2000)*, volume 1862 of *LNCS*, pages 1–17. Springer, 2000.

An investigation into the notion of the reserve set of an ASM, exploring the ideas of adding structure within the reserve (such as the hereditarily finite sets of [41]) and the non-determinism of importing new elements.

39. A. Blass and Y. Gurevich. New Zero-One Law and Strong Extension Axioms. *Bulletin of EATCS*, 72:103–122, October 2000.

A formulation of Shelah's proof of a zero-one law for the choiceless polynomial time variant of ASMs [41].

40. A. Blass and Y. Gurevich. Abstract State Machines Capture Parallel Algorithms. Technical Report MSR-TR-2001-117, Microsoft Research, November 2001.

The proof for the sequential ASM thesis [183] is extended to parallel synchronous algorithms.

41. A. Blass, Y. Gurevich, and S. Shelah. Choiceless Polynomial Time. *Annals of Pure and Applied Logic*, 100:141–187, 1999.

The question "Is there a computation model whose machines do not distinguish between isomorphic structures and compute exactly polynomial time properties?" became a central question of finite model theory. The negative answer was conjectured in [176]. A related question is what portion of PTIME can be naturally captured by a computation model (when inputs are arbitrary finite structures). A PTIME version of ASMs is used to capture the portion of PTIME where algorithms are not allowed arbitrary choice but parallelism is allowed and, in some cases, implements choice. Earlier versions appeared as Technical Report CSE-TR-338-97, EECS Department, University of Michigan, 1997, and Technical Report MSR-TR-99-08, Microsoft Research, February 1999. See [169].

42. A. Blass, Y. Gurevich, and S. Shelah. On Polynomial Time Computation Over Unordered Structures. *Journal of Symbolic Logic*, page to appear, 2001.

    A consideration of several algorithmic problems near the border of the known, logically defined complexity classes contained in polynomial time, including the choiceless polynomial time defined in [41].

43. A. Blass, Y. Gurevich, and J. Van den Bussche. Abstract state machines and computationally complete query languages. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 22–33. Springer-Verlag, 2000.

    The use of the choiceless polynomial-time variant of ASMs [41] as a query language for relational databases is explored. Also appears in TIK-Report 87, ETH Zürich, March 2000, 40–65, and as Microsoft Research Technical Report MSR-TR-99-95.

44. 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 [46].

45. 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 [46].

46. 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 [44] and [45] are the original 3 papers which provide a complete ASM formalization of Prolog with all features discussed in the international Prolog standardization working group (WG17 of ISO/IEC JTCI SC22), see [69]. 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 [47, 48, 101]. These three papers were also published in 1990 as IBM Germany Science Center Research Reports 111, 115 and 117 respectively. The refinement technique, used in combination with corresponding methods of proof, is further developed in [100, 102, 83, 27, 28, 73, 88, 105, 89, 106, 291, 275] and became a constituent of the ASM method.

47. E. Börger. A Natural Formalization of Full Prolog. *Newsletter of the Association for Logic Programming*, 5(1):8–9, 1992.

    The paper explains the abstract tree structure and the four ASM transition rules which govern the user-defined core of Prolog. See [46].

48. E. Börger. Dynamische Algebren und Semantik von Prolog. In *E. Börger: Berechenbarkeit, Komplexität, Logik*, pages 476–499. Vieweg, 1992 (3d edition).

    The first textbook definition of ASMs, elaborating notes of a series of lectures on

Semantics of Programming Languages delivered to a summer school organized by Börger in Cortona in 1989. The definition is illustrated with machines operating on standard data structures and by the tree based version [47] of the core Prolog ASM in [44].

49. 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 1988–1994 on specifications of logic programming systems by ASMs.

50. E. Börger. Review of E.W.Dijkstra and C.S.Scholten *Predicate Calculus and Program Semantics* (Springer-Verlag 1989). *Science of Computer Programming*, 23:1–11, 1994.

    Discusses the weakness of identifying the notion of proof with "formal proofs" and furthermore with "formal proofs in a strict format". Critically evaluates the authors' restricted view on the role of formal methods for program design and verification concerns. An abridged version appeared in *Journal of Symbolic Logic* 59 (1994) 673-678.

51. E. Börger (Ed.). *Specification and Validation Methods*. Oxford University Press, 1995.

    The ASM related papers appearing in this volume are [181, 52, 102, 303, 83, 204, 189].

52. 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. For an updated version see [85].

53. 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 detailed and improved in [88].

54. 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.

55. E. Börger. Remarks on the History and some Perspectives of Abstract State Machines in Software Engineering. In W. Aspray, R. Keil-Slawik, and D. L. Parnas, editors, *The History of Software Engineering* , pages 12–17. Dagstuhl Seminar No. 9635, August 1996.

    Survey of the development of the ASM method as of 1996. For an update in 2000 see [60].

56. E. Börger. How to use Abstract State Machines in Software Engineering. In S. Jähnichen, J. Loeckx, D.R. Smith, and M. Wirsing, editors, *Logic for Systems Engineering*, volume 171, pages 5–7. Dagstuhl Seminar, March 3-7 1997.

    The talk which triggered the first two years of work on the Java/JVM ASM project, as a comparative field test of purely declarative (functional or axiomatic) methods and their enhancement within an integrated abstract state-based operational (ASM) framework [106, 107, 108, 109, 110]. See preface to [291].

57. E. Börger. Ten Years of Gurevich's Abstract State Machines. In E. Börger, editor, *Journal of Universal Computer Science*, volume 3(4). Springer-Verlag, 1997.

    Introduction to the first special ASM issue of the J. of Universal Computer Science. This April issue contains [194, 36, 129, 294, 193, 227, 276].
58. E. Börger. JUCS Special ASM Issue. Part II. In E. Börger, editor, *Journal of Universal Computer Science*, volume 3(5). Springer-Verlag, 1997.

    Introduction to the second part of the special ASM issue of the J. of Universal Computer Science. This May issue contains [224, 225, 318, 13, 89, 237, 306].
59. E. Börger. High Level System Design and Analysis using Abstract State Machines. In D. Hutter and W. Stephan and P. Traverso and M. Ullmann, editor, *Current Trends in Applied Formal Methods (FM-Trends 98)*, number 1641 in LNCS, pages 1–43. Springer-Verlag, 1999.

    A general introduction to and survey of the ASM method, including the definition of the ASM concept and an illustration of the main characteristics of the method, a comparison with other well-known system design and analysis approaches, and experimental evidence for the ASM thesis.
60. E. Börger. Abstract State Machines at the Cusp of the Millenium. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 1–8. Springer-Verlag, 2000.

    A brief survey of the history of the development of the ASM method and the current challenges in the field (continuation of [55]).
61. E. Börger (Ed.). *Hardware Design and Validation Methods*. Springer-Verlag, 2000.

    The ASM related paper appearing in this volume is [109].
62. E. Börger. Design for Reuse via Structuring Techniques for ASMs. In R. Moreno-Diaz and B. Buchberger and J-L. Freire, editor, *Computer Aided Systems Theory–EUROCAST 2001*, volume 2178 of *LNCS*, pages 20–35. Springer-Verlag, 2001.

    The composition and structuring concepts for sequential ASMs defined in [103] are used to illustrate a modular high-level definition of the architecture of the Java Virtual Machine, unfolding its language layering and its functional components for loader, verifier, and interpreter. Extracted from [291].
63. E. Börger. Discrete Systems Modeling. In R. A. Meyers, editor, *Encyclopedia of Physical Science and Technology (Vol.4)*, pages 535–546. Academic Press (San Diego), 2001.

    A classification of discrete systems and of methods for their mathematical verification and experimental validation, using ASMs as framework for the taxonomy.
64. E. Börger. Abstract State Machines: A Naturally Universal Computation Model. In P. Mosses, editor, *Proc. FLoC'02 Workshop Action Semantics and Related Semantic Frameworks*, BRICS Series. Department of Computer Science at University of Aarhus, 2002.

    Continuing the work in [54, 59, 111] representative computation models in the literature are characterized as naturally arising special classes of ASMs. Classical automata (Moore-Mealy, Co-Design FSM, Timed FSM, PushDown, Turing, Scott, Eilenberg, Minsky, Wegner), grammar formalisms, tree computation machines, structured programs are covered, as well as system design models like UNITY, B, SCR (Parnas tables), Petri nets, Neural Nets, and logic based systems including CSP, Z, VDM.

65. E. Börger, H. Busch, J. Cuellar, P. Päppinghaus, E. Tiden, and I. Wildgruber. Konzept einer hierarchischen Erweiterung von EURIS. Siemens ZFE T SE 1 Internal Report BBCPTW91-1 (pages 1-43), Summer 1996.

ASMs are proposed to extend the EURIS method for tool supported design of railway related software.

66. E. Börger, A. Cavarra, and E. Riccobene. An ASM Semantics for UML Activity Diagrams. In Teodor Rus, editor, *Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 20-27, 2000 Proceedings*, volume 1816 of *LNCS*, pages 293–308. Springer-Verlag, 2000.

ASMs are used to disambiguate the semantics for activity diagrams in UML, defining a special subclass of ASMs appropriate to modeling such diagrams. As illustration a one-page UML activity diagram definition is given for the ASM model of Occam which appeared in [74]. For a continuation of this work to make semantic features of UML precise see [67].

67. E. Börger, A. Cavarra, and E. Riccobene. Modeling the Dynamics of UML State Machines. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 223–241. Springer-Verlag, 2000.

The work in [66] providing a rigorous semantics for basic UML features is extended by an ASM definition of the dynamic semantics of UML state machines. These machines integrate statecharts with the UML object model. A rational reconstruction is given for the event driven run to completion scheme of UML (including the sequential entry/exit actions, the concurrent internal activities, and the event deferring mechanism) and for the concepts of action and durative action. The models make the *semantic variation points* of UML explicit, as well as various ambiguities and omissions in the official UML documents. For an executable version of these models see [116] where also various conflict situations are described which may arise through the concurrent behavior of active objects. This argument has been reconsidered by the same authors in *Solving Conflicts in UML State Machines Concurrent States* presented to the Workshop on Concurrency Issues in UML - UML 2001, Toronto/Canada, and in *A precise semantics of UML State Machines: Making Semantic Variation Points and Ambiguities Explicit* in the Proceedings of the International Workshop on Semantic Foundations of Engineering Design Languages (SFEDL'02) in conjunction with the 5th European Joint Conferences on Theory and Practice of Software (ETAPS'02).

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

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

69. 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 [44, 45, 46] proposed to the International Prolog Standardization Committee as a complete formal semantics of Prolog. A streamlined version is in [101], representing the definition of the dynamic core of Prolog which has been accepted as the ISO standard [211].

70. 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 an ASM based technique 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. This work is based upon G. Del Castillo's Tesi di Laurea "Descrizione Matematica dell'Architettura Parallela APE100", Università di Pisa, academic year 1993/94.

71. E. Börger and G. Del Castillo and 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 model is refined in [70] to an ASM processor model.

72. 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 [44, 45, 46]. 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.

73. E. Börger and I. Durdanovic. 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 [82]. Sharpens the refinement method of [100] 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 [74], 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 [70, 88]).

74. E. Börger, I. Durdanovic, 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.

Improving upon the parse tree determined ASM in [190], a truly concurrent ASM model of Occam is defined as basis for a proven to be correct, smooth transition to the Transputer Instruction Set architecture. This model is stepwise refined, in a proven to be correct way, providing: (a) an asynchronous implementation of synchronous channel communication, (b) its optimization for internal channels, (c) the sequential implementation of processors using priority and time–slicing. See [73] for the extension of this work to cover the compilation to Transputer code.

75. 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.

After Börger's lectures on ASMs at the University of Paderborn in the early summer of 1993, Glässer suggested to provide an ASM model for the Parallel Virtual machine (PVM [151], the Oak Ridge National Laboratory software system that serves as a general purpose environment for heterogeneous distributed computing). The model in this paper defines PVM at the C–interface, at the level of abstraction which is tailored to the programmer's understanding. Cf. the survey *An abstract model of the parallel virtual machine (PVM)* presented at *7th International Conference on Parallel and Distributed Computing Systems* (PDCS'94), Las Vegas/Nevada, 5.-9.10.1994. See [76] for an elaboration of this paper.

76. 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 method are explained by showing how one can develop from scratch an easily understandable and transparent ASM model for PVM [151], the widespread virtual architecture for heterogeneous distributed computing.

77. E. Börger and U. Glässer. Abstract State Machines 2001: New Developments and Applications. In Egon Börger and U. Glässer, editors, *Journal of Universal Computer Science*, volume 7(11), pages 914–917. Springer-Verlag, 2001.

Introduction to the third special ASM issue of JUCS, with papers selected from those submitted after the International ASM'2001 Workshop held in Las Palmas. This issue contains [196, 275, 290, 111, 141, 148, 278].

78. E. Börger and U. Glässer. Abstract State Machines Workshop 2001. In R. Moreno-Diaz and A. Quesada-Arencibia, editors, *Formal Methods and Tools for Computer Science*, pages 212–304. IUCTC Universidad de Las Palmas de Gran Canaria, 2001.

Abstracts of talks presented to the International ASM'2001 Workshop held in Las Palmas de Gran Canaria from February 13-19, 2001, as part of Eurocast 2001. See [77].

79. 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 [80].

80. 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 [79] by including the treatment of variable assignments and of value propagation by ports. [271, 268] extend the VHDL model to analog VHDL and to Verilog.

81. E. Börger and R. Gotzhein. The Light Control Case Study. *J. Universal Computer Science*, 6(7):580–585, 2000.

    The introductory pages 580-585 present the requirements engineering case study, discussed during a Dagstuhl Seminar on Requirements Engineering [84], and a synopsis of the six solutions published in the journal issue. For the solution which uses ASMs see the comment to [94].

82. 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 [36, 179, 182, 181, 185, 187, 184, 83, 70, 73, 76] which were used as material for a course on ASMs delivered by the two authors at BRICS, Aarhus, in the summer of 1995.

83. 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. See also [118, 195].

84. E. Börger, B. Hörger, D. Parnas, and D. Rombach. *Requirements Capture, Documentation, and Validation (Seminar No. 99241)*, volume 241. Schloss Dagstuhl, June 1999.

    The Light Control Case Study was proposed to the participants of the seminar to discuss methods to solve requirements engineering problems. See [81] for a detailed exposition of some of the proposed solutions, including [94].

85. E. Börger and J. Huggins. Abstract State Machines 1988-1998: Commented ASM Bibliography. *Bulletin of EATCS*, 64:105–127, February 1998.

    The 1997 version of the annotated bibliography of papers which deal with or use ASMs. An update of [52].

86. E. Börger, P. Joannou, and D. Parnas. *Practical Methods for Code Inspection and Documentation (Seminar No. 9720)*, volume 178. Schloss Dagstuhl, May 1997.

87. 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 [101] 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.

88. 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, 1997.

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, July 1996. The specification was worked out in 1994/95 by S. Mazzanti for her Tesi di Laurea *Algebre Dinamiche per il DLX*, Università di Pisa, 1995, supervised by Börger. For a machine-oriented verification of part of this specification using KIV see [153]. For an omission in the proof for the last refinement step see [203]. The specification and proof method has ben applied in [206] to the commercial ARM2 RISC Microprocessor and enhanced in [297] to automatically transform register transfer descriptions of microprocessors into executable ASMs.

89. 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 easily modifiable code out of rigorous ASM *models for requirement specifications*. Shows that the code properties of interest (like correctness, safety, liveness and performance conditions) can be proved at high levels of abstraction by traditional and reusable mathematical arguments which—where needed—can be computer verified. Shows also that the proposed method is appropriate for dealing in a rigorous but transparent manner with hardware-software co-design aspects of system development.
The approach is illustrated by developing a C^{++} program for the production cell case study. The program has been validated by extensive experimentation with the FZI production cell simulator in Karlsruhe and has been submitted for inspection to the Dagstuhl seminar on "Practical Methods for Code Documentation and Inspection" [86]. Source code (the ultimate refinement) for the case study appears in [237]; model checking results for the ASM models appear in [306] and in [256] where an error was detected in a refinement step for the deposit belt, due to an erroneous assumption of symmetry between unloading actions for feedbelt, press and deposit belt. For a PVS verification of the case see [147]. An abstract appeared under the title "The Evolving Algebra Approach to Modular Development of Well Documented Software for Complex Systems. A Case Study: The Production Cell Control Program" in the Proc. DIMACS Workshop on Controllers for Manufacturing and Automation: Specification, Synthesis, and Verification Issues–CONMASSYV, May 1996, DIMACS. The work was part of Mearelli's Tesi di Laurea *Sviluppo Sistematico di un Programma di Controllo per un Impianto di Produzione Robotizzato*, Pisa 1994/95, supervised by Börger.

90. E. Börger, P. Päppinghaus, and J. Schmid. Report on a Practical Application of ASMs in Software Design. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 361–366. Springer-Verlag, 2000.

A report on the successful use of ASMs at Siemens AG (from May 1998 to March

1999) to design and implement the railway process model component of FALKO, a railway timetable validation and construction program.

91. 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 [92].

92. 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 and is obtained combining the concurrent features of the Occam model of [190] with the logic programming model of [47]. Also published as Technical Report TR 1/93 from Dipartmento di Informatica, Università di Pisa, 1993. Improved and extended version of the following two papers by the same authors: *Logical Operational Semantics of Parlog. Part I: And-Parallelism* in H. Boley and M. M. Richter (Eds.): Processing Declarative Knowledge (Springer Lecture Notes in Artificial Intelligence vol.567 (1991), pages 191-198). *Logical Operational Semantics of Parlog. Part II: Or-Parallelism* in A. Voronkov (Ed.): Logic Programming (Springer Lecture Notes in Artificial Intelligence vol. 592 (1992), pages 27-34). For an extension to Pandora see [265].

93. E. Börger and E. Riccobene. Logic + Control Revisited: An Abstract Interpreter for Gödel Programs. In G. Levi, editor, *Advances in Logic Programming Theory*, pages 231–154. Oxford University Press, 1994.

    Develops a simple ASM interpreter for Gödel programs. This interpreter abstracts from the deterministic and sequential execution strategies of Prolog [100] 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.

94. E. Börger, E. Riccobene, and J. Schmid. Capturing Requirements by Abstract State Machines: The Light Control Case Study. *Journal of Universal Computer Science*, 6(7):597–620, 2000.

    ASMs are applied to the Light Control Case Study discussed during a Dagstuhl Seminar on Requirements Engineering [84]. A ground model is defined which captures the informal requirements as far as possible and documents their ambiguity and incompleteness. The ground model is then refined into a form directly executable by AsmGofer [277].

95. E. Börger and D. Rosenzweig. An Analysis of Prolog Database Views and their Uniform Implementation. ISO/IEC JTCI SC22 WG17 Prolog Standardization Document 80, National Physical Laboratory, Teddington, Middlesex, England, 1991.

    A mathematical analysis of the Prolog database views defined in [72]. The analysis is derived by stepwise refinement of the stack model for Prolog from [100]. It leads to the proposal of a uniform implementation of the different views which discloses the tradeoffs between semantic clarity and efficiency of database update view implementations. Also issued as Research Report CSE-TR-89-91 by the EECS Dept., University of Michigan, Ann Arbor.

96. 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 [45] 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 [98] and improved in [100].

97. 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 [44] and the stack implementation of the tree based model of [45] by a pure tree model for Prolog. See also [47, 48], which is the basis for [101] where a mistake in the treatment of the *catch* built-in predicate is corrected.

98. 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 [96] 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 [100]. Also issued as Technical Report CSE-TR-88-91 from EECS Dept, University of Michigan, Ann Arbor, Michigan, 1991.

99. 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 ASM based definition allows a logico–mathematical analysis, rationale and criticism of various proposals made for implementations of these predicates (in particular of *setof*) in current Prolog systems. Foundational companion to section 5, on solution collecting predicates, in [101]. Also issued as *Prolog. Copenhagen papers 2*, ISO/IEC JTC1 SC22 WG17 Standardization report no. 105, National Physical Laboratory, Middlesex, 1993, pp. 33-42.

100. 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, 1995.

The successive refinement method introduced for ASMs in [44, 45, 46] is applied to provide a hierarchy of models as a mathematical basis for constructing provably correct compilers from Prolog to WAM. Various refinement steps take care of different distinctive features ("orthogonal components") of WAM making the specification as well as the correctness proof modular and extendible; examples of such extensions are found in [28, 27, 102, 13, 227]. An extension of this work to an imperative language with parallelism and non determinism has been provided in [73] and is further developed in [107, 110, 291]. See [262, 276] for machine checked versions of the correctness proofs for the refinement steps. Preliminary versions appeared in [96, 98] and as Research Report TR-14/92, Dipartimento di Informatica, Università di Pisa, 1992.

101. E. Börger and D. Rosenzweig. A Mathematical Definition of Full Prolog. *Science of Computer Programming*, 24:249–286, 1995.

An abstract ASM specification of the semantics of Prolog, rigorously defining the international ISO 1995 Prolog standard by stepwise refinement. Revised and final version of [44, 45, 69, 97, 47, 48]. 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.

102. 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 specification and correctness proof, for compiling Prolog programs to the WAM [100], to CLP($\mathcal{R}$) and the constraint logical arithmetical machine (CLAM) developed at IBM Yorktown Heights. For full proofs, see R. Salamone, "Una Specifica Astratta e Modulare della CLAM (An Abstract and Modular Specification of the CLAM)", Tesi di Laurea, supervised by Börger at Università di Pisa, Italy, academic year 1992/93, pp.113.

103. E. Börger and J. Schmid. Composition and Submachine Concepts for Sequential ASMs. In P. Clote and H. Schwichtenberg, editors, *Computer Science Logic (Proceedings of CSL 2000)*, volume 1862 of *LNCS*, pages 41–60. Springer-Verlag, 2000.

Structuring concepts for sequential composition and iteration, parameterization, and encapsulation in ASMs are defined. The concept of recursive submachines has been developed for its use in [291] to provide a modular definition of the statics and the dynamics of Java and of the JVM architecture [62] which can be naturally refined to an executable model, namely written in AsmGofer [277].

104. 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 [44, 45, 46] through extending unifications by constraint systems. This extension was the starting point for the extension of [100] in [26]. A preliminary version of this was issued as IBM Germany IWBS Report 144, 1990.

105. E. Börger and P. Schmitt. A Description of the Tableau Method Using Abstract State Machines. *J. Logic and Computation*, 7(5):661–683, 1997.

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

106. 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*, number 1523 in 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. Streamlined, corrected, and completed in [291]. An extended abstract has been presented by Börger to the IFIP WG 2.2 (University of Graz, 22.-26.9.1997) and by Schulte under the title *Modular Dynamic Semantics of Java* to the Workshop on Programming Languages (Ahrensdorp, FEHMARN island, September 25, 1997), see

University of Kiel, Dept. of CS Research Report Series, TR *Arbeitstagung Programmiersprachen* 1997. An independently developed Java model using ASMs and Montages was published later as technical report in [304]. For an ASM model of Java which is geared to the analysis of the concurrency features see [192].

107. E. Börger and W. Schulte. Defining the Java Virtual Machine as Platform for Provably Correct Java Compilation. In L. Brim and J. Gruska and J. Zlatuska, editor, *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, MFCS'98, Brno, Czech Republic*, number 1450 in LNCS. Springer, August 1998.

A definition of the Java Virtual Machine, along with a provably correct compilation scheme for Java programs to the JVM, based on the ASM semantics for Java presented in [106]. Streamlined, corrected, and completed in [291]. Full version appears as Technical Report, Universität Ulm, Fakultät für Informatik, Ulm, Germany, 1998.

108. E. Börger and W. Schulte. Initialization Problems for Java. *Software – Concepts and Tools*, 20(4), 1999.

Using the models in [106, 107] and reporting results of experiments with current implementations of the JVM it is shown that the treatment of initialization of classes and interfaces in Java and in the Java Virtual Machine do not match, afflicting the portability of Java programs. It is shown that concurrent initialization may deadlock and that various Java compilers violate the initialization semantics through standard optimization techniques.

109. E. Börger and W. Schulte. Modular Design for the Java VM architecture. In E. Börger, editor, *Architecture Design and Validation Methods*, pages 297–357. Springer, 2000.

Provides a modular definition of the Java VM architecture, at different layers of abstraction. The layers partly reflect the layers made explicit in the specification of the Java language in [106]. The ASM model for JVM defined here and the ASM model for Java defined in [106] provide a rigorous framework for a machine independent mathematical analysis of the language and of its implementation, including compilation correctness conditions, safety and optimization issues. Streamlined, corrected, and completed in [291].

110. E. Börger and W. Schulte. A Practical Method for Specification and Analysis of Exception Handling: A Java/JVM Case Study. *IEEE Transactions on Software Engineering*, 26(10):872–887, October 2000.

ASM models for exception handling in Java and the Java Virtual Machine (JVM) are given, along with a compilation scheme for Java to JVM code. It is proven that corresponding runs of the Java and JVM throw the same exceptions with equivalent effect. A different proof is offered in [291].

111. E. Börger and D. Sona. A Neural Abstract Machine. *Journal of Universal Computer Science*, 7(11):1007–1024, 2001.

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

112. D. Bowen. Implementation at Quintus of Börger's Prolog ASM. Personal Communication to Börger at Quintus in Palo Alto, 5.11.1990.

The four ASM rules which constitute the core for user-defined predicates in Börger's Prolog model [69, 47] have been implemented, making use of the code available at Quintus to compute the abstract functions which appear in that model, in particular the function *unify*, and the function *procdef* which for a given goal (literal) and a given program yields the ordered set of alternatives the program offers for resolving the goal.

113. Manfred Broy, Stephan Merz, and Katharina Spies. The RPC Memory Case Study: A Synopsis. In Manfred Broy and Stephan Merz and Katharina Spies, editor, *Formal Systems Specification – The RPC-Memory Specification Case Study*, number 1169 in LNCS. Springer, August 1996.

For an ASM solution of the case study see [205].

114. W. Burgard, A. B. Cremers, D. Fox, M. Heidelbach, A. M. Kappel, and S. Lüttringhaus-Kappel. Knowledge-enhanced CO-monitoring in Coal Mines. In *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-AIE)*, volume Proc.96, 1996.

Extends the ASM interpreter of [219] by modules, which can be executed in parallel, so that distributed processes can be represented which are synchronized via stream communication. Also a graphical visualization is added, needed for industrial applications of the system in a time- and security-critical coal mining application reported in the paper. Available at `http://www.informatik.uni-bonn.de/~angelica/publications.html`.

115. S. Cater and J. Huggins. An ASM Dynamic Semantics for Standard ML. Technical Report CPSC-1999-2, Kettering University, October 1999.

ASMs are used to provide dynamic semantics for the functional programming language Standard ML. An extended abstract appears in Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, eds., "Abstract State Machines: Theory and Applications", Springer LNCS 1912, 2000, 203-222, and in TIK-Report 87, ETH Zürich, March 2000, 68–99.

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

The thesis which was supervised by Börger and Riccobene studies the use of ASMs to rigorously support semi-formal specification techniques as they are used in industrial practice, with a focus on UML notations and concepts. In addition to the work which has been published in [66, 67] a simulator for UML state machines has been developed using AsmGofer [277].

117. A. Cavarra and E. Riccobene and A. Zavanella. A formal model for the parallel semantics of P3L. In J. Carroll and E. Damiani and H. Haddad and D. Oppenheim, editor, *Proc. of the 2000 ACM Symposium on Applied Computing*, volume 2 of *LNCS*, pages 804–812. ACM Press, March 2000.

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

118. J. Cohen and A. Slissenko. On Verification of Refinements of Asynchronous Timed Distributed Algorithms. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 34–49. Springer-Verlag, 2000.

A study of the role of timing constraints for proving the correctness of refinements of distributed asynchronous algorithms with continuous time, specified as distributed ASMs. The ASM investigation of Lamport's Bakery Algorithm in [83] is used as a case study. Also appears in TIK-Report 87 of ETH Zürich, March 2000, 100–114.

119. K. Compton, Y. Gurevich, J. Huggins, and W. Shen. An Automatic Verification Tool for UML. Technical Report CSE-TR-423-00, EECS Department, University of Michigan, 2000.

Using the ideas developed in [66, 67, 116], ASMs are used to give semantics for

UML state machines, as a basis for constructing an automated tool for verifying properties of UML state machines. An extended abstract appears as "A Semantic Model for the State Machine in the Unified Modeling Language" in G. Reggio, A. Knapp, B. Rumpe, B. Selic, and R. Wieringa, eds., "Dynamic Behaviour in UML Models: Semantic Questions", Workshop Proceedings, UML 2000 Workshop, Ludwig-Maximilians-Universität München, Institut für Informatik, Bericht 0006, October 2000, 25-31.

120. F. DaCruz. *Kermit: A File Transfer Protocol.* Digital Press, 1987.

This book served as basis for the ASM specification and correctness proof in [204].

121. O. Dahl, E. Dijkstra, and C. Hoare. *Structured Programming.* Academic Press, 1972.

122. G. Del Castillo. Towards Comprehensive Tool Support for Abstract State Machines. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *Applied Formal Methods — FM-Trends 98*, volume 1641 of *LNCS*, pages 311–325. Springer-Verlag, 1999.

A description of the ASM Workbench, an integrated environment for various ASM tools, see [123]. Another description appears under the title *The ASM Workbench: an Open and Extensible Tool Environment for Abstract State Machines* in [161, 139-154].

123. G. Del Castillo. *The ASM Workbench. A Tool Environment for Computer-Aided Analysis and Validation of Abstract State Machine Models.* PhD thesis, Universität Paderborn, 2001.

Published in: HNI-Verlagsschriftenreihe Vol.83, pp.IV+212. The main contribution of the thesis, supervised by Börger and Glässer, is the definition of the ASM-based specification language ASM-SL and a tool architecture—the ASM Workbench—based on ASM-SL. The tool environment includes basic functionalities such as parsing, abstract syntax trees, type checking, pretty printing, etc., and in particular a transformation of ASMs into FSMs which can be model checked using SMV, see [128]. In the thesis a case study from the domain of automated manufacturing is treated, namely the distributed control for a material flow system. The ASM Workbench has been extensively used for testing purposes in the FALKO project at Siemens [90]. It has been used in [251] to provide an executable semantics for UML.

124. 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. The definition proceeds by stepwise refinement and leads to the design of a simple virtual machine architecture as a basis for a sequential implementation of the EAM. A preliminary version appeared under the title *Specification and Design of the EAM (EAM - Evolving Algebra Abstract Machine)* as Technical Teport tr-rsfb-96-003, Paderborn University, 1996.

125. G. Del Castillo and U. Glässer. Computer-Aided Analysis and Validation of Heterogeneous System Specifications. In F. Pichler, R. Moreno-Diaz, and P. Kopacek, editors, *Computer Aided Systems Theory: Proceedings of the 7th International Workshop on Computer Aided Systems Theory (EUROCAST'99)*, volume 1798 of *LNCS*, pages 55–79. Springer, 2000.

ASMs are proposed as a method for combining heterogeneous specifications. As a case study, Petri-net and SDL specifications of a material flow system are combined via ASMs and validated using SMV [128].

126. G. Del Castillo and W. Hardt. Towards a Unified Analysis Methodology of HW/SW Systems based on Abstract State Machines: Modelling of Instruction Sets. In *Proceedings of the GI/ITG/GMM Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, 1998.

Extending the processor description technique from [70], ASMs are used for high-level analysis of hardware/software systems. The authors show how to model instruction sets using ASMs and to instrument such models to collect data for evaluating design alternatives. Experimental results appear in [127].

127. G. Del Castillo and W. Hardt. Fast Dynamic Analysis of Complex HW/SW Systems based on Abstract State Machine Models. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE'98) (March 15-18, Seattle, Washington)*, pages 77–81, 1998.

Provides experimental results for [126].

128. G. Del Castillo and K. Winter. Model Checking Support for the ASM High-Level Language. In S. Graf and M. Schwartzbach, editors, *Proceedings of the 6th International Conference TACAS 2000*, volume 1785 of *LNCS*, pages 331–346. Springer-Verlag, 2000.

Extending [306], the authors introduce an interface from the ASM Workbench to the SMV model checking tool, based on an ASM-to-SMV transformation. Previously appeared as Universität-GH Paderborn Technical Report TR-RI-99-209. For an extension see [307, 309]. For an experiment with this interface see [125].

129. 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, Schönhage's storage modification machines are equivalent to unary basic ASMs without external functions. The unary restriction can be removed if the storage modification machines are equipped with a pairing function in an appropriate way.

130. 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.

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

131. 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 a proposed extension to the language of ASMs.

132. B. DiFranco. Specification of ISO SQL using Montages. Master's thesis, Università di l'Aquila, 1997. Tesi di Laurea, in Italian.

133. V. Di Iorio, R. Bigonha, and M. Maia. A Self-Applicable Partial Evaluator for ASM. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract*

State Machines – ASM 2000, International Workshop on Abstract State Machines, Monte Verità, Switzerland, Local Proceedings, number 87 in TIK-Report, pages 115–130. ETH Zürich, March 2000.

A partial evaluator for ASMs is described which is self-applicable. The use of such a tool for compiler generation and techniques for describing language semantics suitable for partial evaluation are discussed. Implementation details are in *An ASM Implementation of a Self-Applicable Partial Evaluator* by V. Di Iorio and R. Bigonha, Technical Report LLP-004-2000 of Programming Languages Laboratory, DCC, Universidade Federal de Minas Gerais. Extends the work of [185].

134. E.W. Dijkstra. Structure of the T.H.E. Multiprogrammming System. *Communications of ACM*, 11:341–346, 1968.
135. A. Dold. A Formal Representation of Abstract State Machines Using PVS. Verifix Technical Report Ulm/6.2, Universität Ulm, July 1998.

A technique for formally representing ASMs using the automated verification system PVS is described, along with generic PVS theories which define refinement relations between ASMs. An application to *Representing the Alpha Processor Family using PVS* by same author appears as Verifix/Uni Ulm/4.1, University of Ulm November 1995.

136. A. Dold, T. Gaul, V. Vialard, and W. Zimmerman. ASM-Based Mechanized Verification of Compiler Back-Ends. In U. Glässer and P. Schmitt, editors, *Proceedings of the Fifth International Workshop on Abstract State Machines*, pages 50–67. Magdeburg University, 1998.

Using techniques from [318], an approach is described for mechanically proving the correctness of back-end rewrite system (BURS) specifications where source and target languages are described by ASMs. The approach can be used in conjunction with BURS-based back-end compiler generators. PVS proof strategies are defined for an automatic verification of BURS rules. Similar aspects are treated by A. Dold and T. Gaul and W. Zimmerman in *Mechanized Verification of Compiler Back-Ends* in the Proc. of the International Workshop on Software Tools for Technology Transfer (STTT'98), Aalborg, Denmark, July 12-13, 1998.

137. A. Durand. Modeling Cache Coherence Protocol – A Case Study with FLASH. In U. Glässer and P. Schmitt, editor, *Proceedings of the Fifth International Workshop on Abstract State Machines*, pages 111–126. Magdeburg University, 1998.

During his research stay in Pisa in 1997/98, upon Börger's suggestion Durand investigated the cache coherence protocol in the Stanford FLASH multiprocessor system for which he provides a high-level specification and correctness proofs related to data consistency. For a model checking verification of the model using SMV see [307].

138. I. Durdanović. From Operational Specifications to Real Architectures. Draft of PhD Thesis (NEC Research Institute Princeton), March 2, 2000.

This PhD project, supervised by Börger, continues the ideas presented in [124]. An ASM Virtual Architecture is defined as basis for a comprehensive ASM tool environment. The developed base system contains an ASM parser and a compiler into ASM/VA code which is a form of high-level C++ programs whose actual refinement into C++ is supported by programs in a C++ library.

139. R. Eschbach. A Termination Detection Algorithm: Specification and Verification. In J. Wing, J. Woodcock, and J. Davies, editors, *Proceedings of FM'99 (Vol.II)*, number 1709 in LNCS, pages 1720–1737. Springer-Verlag, 1999.

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

140. R. Eschbach, U. Glässer, R. Gotzhein, and A. Prinz. On the Formal Semantics of SDL-2000: A Compilation Approach Based on an Abstract SDL Machine. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 242–265. Springer-Verlag, 2000.

An overview of the semantics of SDL-2000, whose complete and final definition which uses ASMs appears in [212]. A simplified language SPL is defined and described using ASMs to point out some of the unique features of the semantics of SDL-2000. Also in TIK-Report 87, ETH Zürich, March 2000, 131–151.

141. R. Eschbach, U. Gässer, R. Gotzhein, M. v. Löwis, and A. Prinz. Formal Definition of SDL-2000—Compiling and Running SDL Specifications as ASM Models. *Journal of Universal Computer Science*, 7(11):1025–1050, 2001.

Contains the most recent and detailed survey of the SDL-2000 formal semantics definition [212] that has been accepted in 2000 by ITU-T, the international standardization body for telecommunication. The focus of this survey is on the dynamic semantics, where ASMs have been applied as the underlying framework. In particular, the SDL Abstract Machine (SAM) model including real time, the definition of SAM programs, and their execution by the SDL Virtual Machine (SVM) (SDL-to-ASM compiler and further tool support) are presented.

142. L. M. G. Feijs and H. B. M. Jonkers. *Formal Specification and Design*. Cambridge University Press, 1992.

Volume 35 of Cambridge Tracts in Theoretical Computer Science.

143. L. M. G. Feijs, H. B. M. Jonkers, C. P. J. Koymans, and G. R. Renardel de Lavalette. Formal Definition of the Design Language COLD-K. In *ESPRIT Document METEOR/t7/PRLE/7*, April 1987.

Final update in August 1989.

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

The paper describes an ASM based prototype system, in the spirit of active databases, for specifying electronic commerce applications. An extensible database model called "evolving databases" (EDB) is defined based upon ASMs. It is applied to capture in a rigorous transparent way the state changes involved in electronic commerce negotiations, concerning the traded products, the negotiators, their orders, and the laws accepted as basis for the particular negotiation. See [1].

145. 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 [224] 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 [224]. In addition, the GemMex 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.

146. M. C. Gaudel. *Génération et Preuve de Compilateurs Basées sur une Sémantique Formelle des Langages de Programmation*. Thèse, L'Institut National Polytechnique de Lorraine, France, 1980.

The work to which usually the idea is attributed to use Tarski structures as most general notion of states. See [254].

147. A. Gargantini and E. Riccobene. Encoding Abstract State Machines in PVS. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 303–322. Springer-Verlag, 2000.

A framework for automatic translation from ASM to PVS is presented. Following a suggestion by Börger, the ASM specification of the Production Cell problem [89] is used as a case study. Also appears in TIK-Report 87, ETH Zürich, March 2000, 152–173.

148. A. Gargantini and E. Riccobene. ASM-Based Testing: Coverage Criteria and Automatic Test Sequence Generation. *Journal of Universal Computer Science*, 7(11):1051–1068, 2001.

ASMs are used for testing purposes, defining adequacy criteria measuring the coverage achieved by a test suite, and determining whether sufficient testing has been performed. An algorithm is defined to generate from ASMs test sequences with desired coverage, exploiting the counter example generation of SMV.

149. 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 is derived directly from the original manufacturer's handbook. The specification omits certain less-used instructions and VAX compatibility parts.

150. T. Gaul, A. Heberle, and W. Zimmermann. An ASM Specification of the Operational Semantics of MIS. Verifix Working Paper Verifix/UKA/3, University of Karlsruhe, 1998.

An ASM specification of MIS, an intermediate programming language used in the Verifix project for provably correct compilation to the DEC-Alpha microprocessor [149].

151. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

A high-level model of the system described in this book has been developed in [75, 76].

152. F. Giannuzzi. Studi di un metodo per la derivazione dei casi di test da specifiche ASM. Tesi di laurea, Universitá di Pisa, July 2001.

Studies the derivation of test cases from ASM specifications, illustrating an application of the cause-effect-graph method for the Production Cell ASM [89]. Supervised by Bertolino and Börger.

153. M. Giese, D. Kempe, and A. Schönegge. KIV zur Verifikation von ASM-Spezifikationen am Beispiel der DLX-Pipelining Architektur. Interner Bericht 16/97, Universität Karlsruhe, 1997.

The Karlsruhe Interactive Verifier (KIV system) is used for the formal verification of the ASM specification of the DLX pipelining architecture in [88]. Details of the verification and estimates of the amount of work required are given, along with several minor shortcomings of the original specification. Two additions to the KIV system are described which were designed in the course of this case study.

154. 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.

155. 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.

156. U. Glässer. ASM Semantics of SDL: Concepts, Methods, Tools. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz, editors, *Proc. of the 1st Workshop of the SDL Forum Society on SDL and MSC*, volume Informatik-Berichte 104 (ISSN 0863-095), pages 271–280. Humboldt-Universität Berlin, 1998.

 Proposal to the SDL Forum to use ASMs for a definition of the semantics of SDL which is abstract but through its operational character is apt to be transformed to an executable model. Detailed in [160].

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

 Contains a systematic treatment of the work started in [160] providing ASM models for the dynamic semantics of SDL. Completed in [212], see [141] for a survey.

158. U. Glässer, R. Gotzhein, and A. Prinz. Towards a New Formal SDL Semantics Based on Abstract State Machines. In G. v. Bochmann, R. Dssouli, and Y. Lahav, editors, *SDL'99 - The Next Millenium, Proceedings of the 9th SDL Forum*, pages 171–190. Elsevier Science B.V., 1999.

 Based upon the idea proposed in [160], ASMs are applied to formally define the behavior model of a sample SDL-2000 specification. See also "SDL Formal Semantics Definition" by the same authors, published as University of Paderborn TR SFBR-99-065, June 1999. See the completion of the work in [212] and the survey [141].

159. U. Glässer and Y. Gurevich and M. Veanes. High-Level Executable Specification of the Universal Plug and Play Architecture. In *Proceedings of 35th Hawaii International Conference on System Sciences -2002*, pages 1–10. IEEE, 2002.

 An ASM specification of the Universal Plug and Play (UPnP) architecture for peer-to-peer network connectivity of intelligent devices. A more detailed version appeared in June 2001 as Microsoft Research technical report MSR-TR-2001-59 under the title "Universal Plug and Play Models".

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

 A formal semantic model of Basic SDL-92 – according to the *ITU-T Recommendation Z.100* – is defined in terms of an abstract SDL machine based on the concept of a *multi-agent real-time ASM*. The resulting interpretation model is not only mathematically precise but also reflects the common understanding of SDL in a direct and intuitive manner; it provides a *concise* and *understandable* representation of the complete dynamic semantics of Basic SDL-92. Moreover, the model can easily be *extended* and *modified*. The article considers the behavior of channels, processes and timers with respect to signal transfer operations and timer operations. Continuation of this work and merging it with work by Gotzhein and by Prinz [158, 157, 261] led to the ITU-T standard definition of SDL-2000 [212, 140].

161. U. Glässer and P. Schmitt. Proceedings of the Fifth International Workshop on Abstract State Machines. Technical Report GI Jahrestagung 1998, Otto-von-Guericke-Universität Magedeburg, 1998.

Extended abstracts of the talks presented to the workshop which was organized as part of the 28th Annual Conference of the German Computer Science Society (GI Jahrestagung). See [314, 282, 235, 136, 200, 299, 137, 31, 122].

162. 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 [190, 91]. As illustration models are given for the Chemical Abstract Machine and the π-calculus. See [181] for a more general definition of the notion of distributed ASM runs.

163. 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.

164. W. Goerigk, A. Dold, T. Gaul, G. Goos, A. Heberle, F. W. von Henke, U. Hoffmann, H. Langmaack, H. Pfeifer, H. Ruess, and W. Zimmermann. Compiler Correctness and Implementation Verification: The Verifix Approach. In P. Fritzson, editor, *International Conference on Compiler Construction*, volume Proceedings of the Poster Session of CC'96, IDA Technical Report LiTH-IDA-R-96-12, Linköping/Sweden, 1996.

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

165. G. Goos, A. Heberle, W. Löwe, and W. Zimmermann. On Modular Definitions and Implementations of Programming Languages. In Y. Gurevich, P. Kutter,

M. Odersky, and L. Thiele, editors, *Abstract State Machines – ASM 2000, International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, number 87 in TIK-Report, pages 174–208. ETH Zürich, March 2000.

A formal composition and refinement (correct implementation) mechanism for state-transition systems is presented which exploits the abstract syntax of programs. Applications are made to language semantic definitions using ASMs. Montages [224] is characterized as a set of parameterized ASMs.

166. 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, in the context of a graphical object-oriented data model design language, the operational semantics of object creation, of overriding and dynamic binding, and of inheritance at the type level (type specialization) and at the instance level (object specialization). Issued also as technical report MooD-TR 90/02 at Technische Universität Wien on December 20, 1990. See [284].

167. G. Goos and W. Zimmermann. Verifiying Compilers and ASMs. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 177–202. Springer-Verlag, 2000.

ASMs are used to describe verifying compilers: compilers which verify the correctness of their generated code.

168. E. Grädel and Y. Gurevich. Metafinite Model Theory. *Information and Computation*, 140(1):26–81, 10 January 1998.

Computer systems, *e.g.* databases, are not necessarily finite because they may involve for example arithmetic. Motivated by such computer science challenges and by ASM applications, metafinite structures, as they typically appear in ASM states, are defined and finite model theory is extended to metafinite models. An early version has been presented under the title *Towards a Model Theory of Metafinite Structures* to the Logic Colloquium 1994, see the abstract in the *Journal of Symbolic Logic*. An intermediate version appeared in *Logic and Computational Complexity, Selected Papers*, Springer LNCS 960, 1995, 313-366.

169. E. Grädel and M. Spielmann. Logspace Reducibility via Abstract State Machines. In J. Wing, J. Woodcock, and J. Davies, editors, *Proceedings of FM'99 (Vol.II)*, number 1709 in LNCS, pages 1738–1757. Springer-Verlag, 1999.

ASMs are used to investigate logspace reducibility among structures, capturing the choiceless fragment of logspace. A continuation of [41]. See also [287].

170. I. Graham. *The Transputer Handbook*. Prentice-Hall, 1990.

Together with [209, 210], this book served as basis for the ASM model developed for the Transputer in [73].

171. W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Conformance Testing with Abstract State Machines. Technical Report MSR-TR-2001-97, Microsoft Research, October 2001.

See [17].

172. 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) covering sequential ASMs. This language is then extended to a multi-modal language FLEA' and it is sketched how one can transfer the axioms of the logic MLCM to FLEA'. MLCM is a Modal Logic of Creation and Modification, a dynamic logic which is incorporated in Jonker's Common Object-Oriented Language for Design COLD [142, 143]. See [290].

173. M. Grosse-Rhode. A Formal Specification Framework for Evolving Algebras. Manuscript Technical University of Berlin, 1996.

Applies some algebraic-categorical composition schemes to ASMs, illustrated on an alternating bit protocol specification.

174. Y. Gurevich. Reconsidering Turing's Thesis: Toward More Realistic Semantics of Programs. Technical Report CRL-TR-38-84, EECS Department, University of Michigan, 1984.

An attempt to reconsider Turing's Thesis, taking into account that resources are bounded. The earliest known paper in which the ideas behind ASMs began to take form. See the continuation in [175].

175. Y. Gurevich. A New Thesis. *Abstracts, American Mathematical Society*, page 317, August 1985.

Following [174], for the first time the ASM Thesis is stated.

176. 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.

Part 2 contains the first small examples for ASMs, drawn from Gurevich's lectures in Semantics of Programming Languages delivered in Pisa in the Spring of 1987.

177. 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.

178. Y. Gurevich. Kolmogorov Machines and Related Issues. *Bulletin of EATCS*, 35:71–82, 1988.

The Kolmogorov-Uspenskii thesis is stated that every computation, performing only one restricted local action at a time, can be viewed as the computation of an appropriate Komogorov-Uspenskii machine.

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

The first tutorial on ASMs. The ASM thesis is stated. 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. A german textbook version of the definition appeared in [48]. For a more elaborate and complete definition see [181].

180. Y. Gurevich. Logic Activities in Europe. *ACM SIGACT News*, 1994.

A critical analysis of European logic activities in computer science. Subsection 4.6 *Mathematics and Pedantics* discusses the separation of different levels of verification in the context of modeling with ASMs.

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

The notion of sequential ASMs defined in [179] is extended to cover distributed computations. A later update *May 1997 Draft of the ASM Guide* appeared as Technical Report CSE-TR-336-97, EECS Dept., University of Michigan.

182. 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.

183. Y. Gurevich. Sequential Abstract State Machines Capture Sequential Algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July 2000.

The notion of "sequential algorithm" is formalized and proved equivalent to the notion of sequential ASMs. The sequential version of the ASM Thesis (proposed in [175, 179] is proved from three basic postulates. An early version appeared under different titles as Microsoft Research Technical Reports MSR-TR-99-09 and MSR-TR-99-65, and in Bulletin of EATCS 67 (February 1999), 93-124.

184. 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 is used to give a succinct dynamic semantics of the C programming language. For a correction of minor errors and omissions see the ERRATA in 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, supervised by Gurevich at the University of Michigan, Ann Arbor, 1995. For an extension to C++ see [303]. For an addition of the statics of C to the model see [207].

185. 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. For an extension of this work see [133].

186. 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 in [202]. The paper experiments with agents that perform instantaneous actions in continuous time at the moment they are enabled. A preliminary version appeared under the title *The Railroad Crossing Problem: An Evolving Algebra Solution* as research report LITP 95/63 of Centre National de la Recherche Scientifique, Paris, and under the title *The Generalized Railroad Crossing Problem: An Evolving Algebra Based Solution* as research report CSE-TR-230-95 of EECS Department, University of Michigan, Ann Arbor, MI. For a further investigation see [21, 22].

187. 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.

188. Y. Gurevich, P. W. Kutter, M. Odersky, and L. Thiele. Abstract State Machines. Theory and Applications. In *Lecture Notes in Computer Science Vol. 1912*, pages X+381. Springer, 2000.

Proceedings of the International Workhop ASM2000 held at Monte Verità, Switzerland, March 2000.

189. 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.

A processor-group membership protocol involving timing constraints is formally specified and verified using distributed ASMs.

190. 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. For an improved (not any more parse tree determined, but truly concurrent) ASM model for Occam and its refinement to a Transputer implementation see [74, 73].

191. Y. Gurevich, W. Schulte, C. Campbell, and W. Grieskamp. AsmL: The Abstract State Machine Language. Version 1.5. Web pages of Foundations of Software Engineering at Microsoft Research, May 21, 2001.

Documentation of the AsmL specification language.

192. Y. Gurevich, W. Schulte, and C. Wallace. Investigating Java Concurrency using Abstract State Machines. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 151–176. Springer-Verlag, 2000.

An ASM specification and verification of Java's model of concurrency, including threads and synchronization. Also in TIK-Report 87, ETH Zürich, March 2000, 227–271, and in University of Delaware Department of Computer & Information Sciences TR 2000-04.

193. 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 the high-level model and of each of the four refinement steps. An updated version of the Technical Reports CSE-TR-249-95 and CSE-TR-327-97 of EECS Department, University of Michigan, Ann Arbor, and of the paper *Formalizing Recovery in Transaction-Oriented Database Systems* of C. Wallace and Y. Gurevich and N. Soparkar, published in S. Chaudhuri and A. Deshpande and R. Krishnamurthy (Eds.): Proceedings of the Seventh International Conference on Management of Data, Tata McGraw-Hill, New Delhi, India, 1995, pages 166-185.

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

A definition of recursive ASMs in terms of distributed ASMs is suggested. A preliminary version appeared as Technical Report CSE-TR-322-96, EECS Department, University of Michigan, Ann Arbor, 1996. For a definition of recursive ASMs in terms of sequential ASMs see [103].

195. Y. Gurevich and D. Rosenzweig. Partially Ordered Runs: A Case Study. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 131–150. Springer-Verlag, 2000.

The ASM investigation [83] of Lamport's Bakery Algorithm is sharpened in terms of partially ordered runs, abstracting from the mapping of moves to linear real-time. Some properties are proved which are useful for reasoning about partially ordered runs. The paper also appeared as technical report in TIK-Report 87, ETH Zürich, March 2000, and in MSR-TR-99-98.

196. Y. Gurevich and N. Tillmann. Partial Updates: Exploration. *Journal of Universal Computer Science*, 7(11):918–952, 2001.

A solution is proposed for the problem of cumulative updates for counters, steps and maps.

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

An ASM specification of the Windows Card Runtime Environment and a verification of certain safety properties.

198. P. Hartel and L. Moreau. Formalizing the Safety of Java, the Java Virtual Machine and Java Card. *ACM Computing Surveys*, 33(4):517–558, 2001.

A review of the literature on formal approaches of Java and its implementation with focus on safety issues and their impact on smart cards. Section 6.2 evaluates the ASM based work in this area [106, 107, 108, 109, 110, 291, 304].

199. A. Heberle. *Korrekte Transformationsphase - der Kern korrekter Übersetzer.* PhD thesis, Universität Karlsruhe, 2000.

The essential results of the thesis (which is written in German) are published in [200, 201].

200. A. Heberle and W. Löwe. On ASM-Based Specification of Programming Language Semantics and Reusable Correct Compilations. In U. Glässer and P. Schmitt, editors, *Proceedings of the Fifth International Workshop on Abstract State Machines*, pages 68–90. Magdeburg University, 1998.

General equivalence-preserving transformations on ASM specifications of programming languages are defined, to be used for the definition of provably correct compilation schemes. An extensible language AL is introduced for specifying dynamic language semantics in a way which facilitates the reuse of verified transformations. Some of the results are from [199].

201. A. Heberle, W. Löwe, and M. Trapp. Safe Reuse of Source to Intermediate Language Compilations. In R. Chillarege, editor, *Proc. 9th. Int. Symp. on Software Reliability Engineering*, 1998. See http://www.chillarege.com/issre/fastabstracts/98417.html. Contains some results of [199].

202. C. Heitmeyer and D. Mandrioli. *Formal Methods for Real-Time Computing*, volume Trends in Software 5. Wiley, 1996.

Extensive study of the Railroad Crossing Problem, proposed as case study for real-time computing and solved using various popular specification and verification methods. For an ASM solution see [186].

203. H. Hinrichsen. Formally Correct Construction of a Pipelined DLX Architecture. Technical Report TR 98-5-1, Darmstad University of Technology, Dept. of Electrical and Computer Engineering, 1998.

In an e-mail to Börger on February 11, 1998, Hinrichsen points out that for a correct handling of the instruction sequence 1. LOAD R1,A, 2. LOAD R2, B, 3. ADD R3,R1,R2, the ADD instruction must be stalled for one clock cycle. This corrects an omission of a hazard case in the last refinement step of [88].

204. 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 [120] is specified and verified using ASMs at several layers of abstraction. This work is part of the author's PhD thesis *Evolving Algebras: Tools for Specification, Verification, and Program Transformation*, pp.IX+91, University of Michigan, Ann Arbor, 1995.

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

Upon Börger's suggestion, Huggins developed an ASM solution to the specification problem proposed by Broy and 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. Other solutions of this problem were published in [113].

206. J. Huggins and D. Van Campenhout. Specification and Verification of Pipelining in the ARM2 RISC Microprocessor. *ACM Transactions on Design Automation of Electronic Systems*, 3(4):563–580, October 1998.

An extended abstract describing a layered ASM specification of the advanced RISC machine processor ARM2, one of the early commerical RISC microprocessors. The method developed in [88] is applied for the layered specification and the correctness proof for the ARM2's pipelining techniques. In [297] this ASM model of the ARM is used to illustrate an approach to automatically transform register transfer descriptions of microprocessors into executable ASMs. A full version of the paper appears as University of Michigan EECS Department Technical Report CSE-TR-371-98. An earlier version appears in *Proceedings of the IEEE International High Level Design Validation and Test Workshop (HLDTV'97)*, November 1997.

207. J. Huggins and W. Shen. The Static and Dynamic Semantics of C. Technical Report CPSC-2000-4, Kettering University, Computer Science Program, 2000.

The ASM for C in [184] is extended to provide both static and dynamic semantics for C, using Montages [224]. An extended abstract appears in Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, eds., *Abstract State Machines – ASM 2000*, International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings, TIK-Report 87, ETH Zürich, March 2000, 272–283. A previous version appears as Kettering University Computer Science Program Technical Report CPSC-1999-1.

208. IEEE Standardization. IEEE Standard VHDL Language Reference Manual. Technical Report Std 1076-1993, IEEE, 1993.

The standard description of the hardware design language VHDL'93 which has been formalized by an ASM ground model in [79, 80].

209. INMOS. *Transputer Instruction Set–A Compiler Writer's Guide*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

INMOS Document 72 TRN 119 05. See the comment to [170].

210. INMOS. *Transputer Implementation of Occam - Communication Process Architecture*. Prentice-Hall, Englewoods Cliff, NJ, 1989.

See comment to [170].

211. ISO. Prolog–Part 1: General Core. ISO Standard Information Technology–Programming Languages ISO/IEC 13211-1, ISO/ICE, January 1995.

212. ITU-T. SDL Formal Semantics Definition. ITU-T Recommendation Z.100 Annex F, International Telecommunication Union, November 2000. This document contains the complete, internationally standardized formal semantics definition of SDL-2000, a design language for the development of distibuted real-time systems in general and telecommunication systems in particular. SDL is industrially

applied in the telecommunications industry, for instance, to the development of UMTS protocols and Intelligent Networks. The dynamic semantics of SDL-2000 is defined using ASMs as the underlying mathematical framework. For further information see http://rn.informatik.uni-kl.de/projects/sdl, for a survey see [141].

213. J. W. Janneck. *Syntax and Semantics of Graphs.* PhD thesis, ETH Zürich, 2000.

Published in: Berichte aus der Informatik, TIK Series Vol.38, Shaker Verlag Aachen (ISBN 3-8265-7688-8), pp.XI+177. The classical networks of stream processing finite state machines (with their notion of network components with input and output ports to communicate among each other) are enriched by ASM state transformations of individual components. The resulting machines are applied to give a uniform rigorous semantics to common visual notations for discrete event systems, together with a prototypical implementation. Illustration by Petri nets.

214. J. Janneck and P. Kutter. Object-based Abstract State Machines. TIK-Report 47, ETH Zürich, 1998.

Proposes to view ASMs as classes attached to objects which communicate only by message passing. Illustration by a class definition for Petri net places and transitions.

215. J. Janneck and P. Kutter. Mapping Automata: Simple Abstract State Machines. TIK-Report 49, ETH Zürich, June 1998.

Mapping automata are defined as ASMs where the state is formed by a single binary function (interpreted as mapping which assigns to every object in the base set U a unary function over objects), and the rules are built from updates of that binary function in the usual way. Using the standard coding of arbitrary structures into the structure of one binary function, the resulting correspondence between mapping automata and ASMs is shown to preserve the desired computational equivalence. Also appears in Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, eds., "Abstract State Machines – ASM 2000", International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings, TIK-Report 87, ETH Zürich, March 2000, 310–325. An implementation in Java is reported in *Object-Based Mapping Automata (Reference Manual)* by J. W. Janneck, TIK-Report 50, ETH Zürich, June 1998. Mapping automata are used in [216] for a description of the semantics of UML statecharts.

216. Y. Jin, R. Esser, and J. W. Janneck. Describing the syntax and semantics of UML Statecharts in a heterogeneous modelling environment. In *Proceedings DIAGRAMS 2002*, 2002.

Based upon the syntactical description of UML statecharts by attributed graphs coming with well-formedness conditions, the mapping automata of [215] are used to describe the semantics of UML statecharts. Compared with the ASM model of UML statecharts in [67], the focus here is on a discussion of transition conflicts.

217. 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 [190, 92] and from the dynamic tree generation appearing in [97, 101]. Then the basic model of context free rules is extended to characterise in a uniform and natural way different context sensitive languages in terms of ASMs. See [241, 242].

218. 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.

219. A. M. Kappel. Implementation of Dynamic Algebras with an Application to Prolog. Diplom thesis, Computer Science Dept., Universität Dortmund, Germany, 1990 (submitted 2.11.1990).

This Diplom thesis was triggered by Börger's lectures on ASM models for Prolog [44, 45], delivered in June 1989 to A. B. C. Cremers' and H. Ganzinger's "Diplomanden-und Doktorandenseminar" at the University of Dortmund. Kappel defines a language for the specification of sequential ASMs and designs an abstract target machine (namely a Prolog program) for executing a class of sequential ASMs, including those of the ASM models for Prolog in [44, 45]. A prototype of the compiler has been implemented in Prolog, all the examples have been tested for Quintus Prolog on a SPARC station 1+ and for LPA Prolog on an IBM PC AT. A short version of the paper appeared in [220], a parallel extension of the interpreter appears in [114].

220. 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.

Short version of [219].

221. A. N. Kolmogorov and V. A. Uspenskii. On the Definition of an Algorithm. *AMS Translations, 2nd Series*, 29:217–245, 1993.

222. P. Kutter. An ASM Macro Language for Sets. TIK-Report 34, ETH Zürich, January 1998.

A small set of simple, generic macros that allow one to manipulate and parametrize sets in ASMs, without changing the semantics given in [181].

223. P.W. Kutter. *Montages - Engineering of Computer Languages.* PhD thesis, ETH Zürich, 2002.

Contains a denotational semantics of XASM [8], an ASM semantics of Montages, an example language illustrating the description of language features found in sequential Java (see [304]).

224. 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. At that same university also the first application of Montages appeared in a Tesi di Laurea [132]. See [11] for an extension of Montages with a finite-state machine model.

225. 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 [224]. The dynamic semantics previously appeared as P. Kutter, "Dynamic Semantics of the Oberon Programming Language", TIK-Report 25, ETH Zürich, Feburary 1997.

226. P. Kutter, D. Schweizer, and L. Thiele. Integrating Domain Specific Language Design in the Software Life Cycle. In *Proceedings of the International Workshop on Current Trends in Applied Formal Methods*, volume 1641 of *Lecture Notes in*

Computer Science, pages 196–212. Springer, 1998.

A report on an industrial case study, applying ASMs and Montages [224] to the design, specification, and implementation of a driver specification language needed in the context of a complex data warehouse problem at Union Bank of Switzerland.

227. K. Kwon. A Structured Presentation 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 [100]. The PhD thesis upon which this paper is based was submitted to Duke University on December 12, 1994, under the title "Towards a Verified Abstract Machine for a Logic Programmming Language with a Notion of Scope", number CS 1994-36, pp.189.

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

Definition of the bakery algorithm to solve the mutual exclusion problem, see also [229]. An ASM analysis of this algorithm appears in [83].

229. L. Lamport. On Interprocess Communication. Part I: Basic Formalism, Part II: Algorithms. *Distributed Computing*, 1:77–101, 1986.

See [228].

230. H. Langmaack. The ProCoS Approach to Correct Systems. *Real-Time Systems*, 13:253–275, 1997.
231. T. Lindner. Task Description. In C. Lewerentz and T. Lindner, editor, *Formal Development of Reactive Systems. Case Study "Production Cell"*, volume 891 of *LNCS*, pages 9–21. Springer-Verlag, 1995.

Description of the Production Cell case study which has been derived from a metal-processing plant in Karlsruhe. The book contains solutions of the problem which use various formal methods. The book inspired to work on an ASM solution of the problem, see [89].

232. 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.

233. A. Lötzbeyer. Simulation of a Steam Boiler. 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 493–499. Springer, 1996.
234. M. Maia and R. Bigonha. An ASM-Based Approach for Mobile Systems. Technical Report LLP-12/99, Programming Language Laboratory, Computer Science Department, Universidade Federal de Minas Gerais, 1999.

Using the Interacting ASM techniques introduced in [235], the authors describe the use of ASMs to specify the semantics of active mobile objects. Mobility is expressed by dynamic changes in the communication topology. An earlier version appears as Technical Report LP 002/99 (same institution).

235. M. Maia, V. Iorio, and R. Bigonha. Interacting Abstract State Machines. In U. Glässer and P. Schmitt, editor, *Proceedings of the Fifth International Workshop on Abstract State Machines*, pages 37–49. Magdeburg University, 1998.

An extended abstract describing an extension to ASMs supporting the interaction of independent ASM agents by means of message passing. Full version appears as M. Maia and R. Bigonha, *Formal Semantics for Interactive Abstract State Machine Language*, Technical Report RT 005/98, Universidade Federal de Minas Gerais, Brazil, 1998. Continued in [234].

236. 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 structured systems with ASMs by means of aggregation and composition. An earlier version appeared under the title "Modeling Rule-Based and Structured Systems with Evolving Algebras" as Technical Report, Freiburg, 1996. For some of the structuring concepts defined here, simpler definitions are given in [103] which are geared to their natural integration into the basic parallelism of multiple simultaneous machine actions of ASMs.

237. 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 ASM specification of the Production Cell described in [89]. For a generation of this code see [278].

238. 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.

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

Thesis supervised by Gurevich. The earliest ASM formalization of a programming language. The semantical description is parse-tree directed, but flat. An extended abstract appeared as Y. Gurevich and J. Morris, “Algebraic Operational Semantics and Modula-2”, in E. Börger, H. Kleine Büning and M. M. Richter, eds., *CSL'87, 1st Workshop on Computer Science Logic*, Springer LNCS 329, 1988, pp. 81-101.

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

241. 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 [217] to grammar formalisms based on Kasper-Rounds logics. See [242].

242. 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 [217] to several other grammar formalisms.

243. 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.

On Börger's suggestion this work extends the rules given in [47] for the user–defined core of Prolog to define the semantics of a hybrid object–oriented Prolog system. The definition covers the central object–oriented features of object creation and deletion, data encapsulation, inheritance, messages, polymorphism and dynamic binding. See [244].

244. B. Müller. *Eine objektorientierte Prolog-Erweiterung zur Entwicklung wissensbasierter Systeme.* PhD thesis, University of Oldenburg, Germany, 1994.

Thesis supervised by Appelrath and Börger. Defines an object oriented extension of Prolog to be applied for the development of knowledge based systems. The semantics is defined (in Chapter 5) as an extension of Börger's Prolog model [47].

245. W. Müller. *Executable Graphics for VHDL-Based Systems Design.* PhD thesis, University of Paderborn, 1996.

Uses ASMs to define the behavioral semantics of PHDL, a pictorial extension of VHDL'93. The ASMs for VHDL defined in [79, 80] are reused.

246. W. Mueller, R. Dömer, and A. Gerstlauer. The Formal Execution Semantics of SpecC. Technical Report TR ICS 01-59, Center for Embedded Computer Systems at the University of California at Irvine, 2001.

Adapting the distributed ASM model of VHDL in [79, 80] and the work in [247], a distributed ASM model for the semantics of SpecC is developed which covers the execution of SpecC behaviors and their interaction with the simulation kernel. This includes wait, waitfor, par, pipe, and try statements.

247. W. Mueller, J. Ruf, D. Hofmann, J. Gerlach, T. Kropf, and W. Rosenstiehl. The Simulation Semantics of SystemC. In *Proc. of DATE 2001*, IEEE CS Press, March 2001.

Adapting the distributed ASM model of VHDL in [79, 80], a distributed ASM model for the semantics of SystemC is developed which covers method, thread, clocked thread behavior, and their interaction with the simulation kernel. Watching statements, signal assignment and wait statements are formalized for version V1.0 of SystemC.

248. M. Müller-Olm. *Modular Compiler Verification. A Refinement-Algebraic Approach Advocating Stepwise Abstraction.* Springer LNCS 1283, 1997.

The author's PhD thesis. The considered language is a sublanguage of Occam with real-time features. See also the PROCOS II Esprit Basic Research 7071 Report MMO 12/3 (1996), University of Kiel: Structuring Code Generator Correctness Proofs by Stepwsies Abstracting the Machine Language's Semantics.

249. Z. Nemeth. Definition of a Parallel Execution Model with Abstract State Machines. *Acta Cybernetica*, 15(3), 2002.

Two ASMs are defined and related by a refinement correctness proof, as preparation for designing and verifying a distributed parallel Prolog execution model.

250. Z. Nemeth and V. Sunderam. A Formal Framework for Defining Grid Systems. In *Proceedings of International Symposium on Cluster Computing and the Grid (CCGrid2002).* IEEE Computer Society Press, 2002.

ASMs are used to define a model for grid systems.

251. I. Ober. More Meaningful UML Models. In *Proceedings of TOOLS.* IEEE Computer Society Press, 2000.

ASMs are used to define an executable semantics for UML which covers real-time aspects. The work is inspired by the ASM model for SDL in [158] and uses the ASM Workbench [123].

252. M. Odersky. Programming with Variable Functions. In *ICFP'98, Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, volume 34 (1) of *ACM SIGPLAN Notices*, pages 105–116, January 1999.

The use of "variable functions" (functions which can be updated at specified points in their domains) is proposed as a method for deriving efficient imperative programs from functional programs. The notion of variable function is drawn from the dynamic functions of ASMs.

253. C. Pahl. Towards an Action Refinement Calculus for Abstract State Machines. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines – ASM 2000, International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, number 87 in TIK-Report, pages 326–340. Swiss Federal Institute of Technology (ETH) Zurich, March 2000.

A refinement calculus for (a reformulation of) ASMs is presented.

254. C. Pair. Types Abstraits et Sémantique Algébrique des Langages de Programmation. Technical Report TR 80-R-011, Centre de Recherche en Informatique de Nancy, 1980.

Apparently the first publication where the idea is formulated that the most general notion of state of computing systems is Tarski's notion of structures. See also [146].

255. B. Pehrson and I. Simon. I: Technology/foundations. In *IFIP 13th World Computer Congress 94*, Elsevier, Amsterdam, the Netherlands, 1994.

Stream C (Evolving Algebras) (pages 377–441), organized by Gurevich, contains short versions of the talks presented to the first international ASM workshop, see [25, 36, 49, 71, 75, 87, 163, 182, 243, 259, 266].

256. C. N. Plonka. Model Checking for the Design with Abstract State Machines. Diplom thesis, CS Department of University of Ulm, Germany, January 2000.

A feasibility study, carried out upon Börger's suggestion at Siemens Research, of model checking ASMs for two industrial case studies, the Production Cell [89] and a statistical multiplexing unit. An error was detected in [89] concerning a refinement step for the deposit belt, due to an erroneous (easily to be repaired) symmetry assumption made during the specification for the unloading actions of feedbelt, press and deposit belt. Due to additional scheduling assumptions, made for the model checking of the Production Cell ASM in [306] to guarantee maximal performance of the model, the mistake had remained undiscovered there.

257. 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.

258. 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.

259. 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. See [290].

260. A. Poetzsch-Heffter. Prototyping Realistic Programming Languages Based On Formal Specifications . *Acta Informatica* , 34:737–772, 1997.

A tool supporting the generation of language-specific software from specifications is presented, enabling in particular the generation and refinement of interpreters based on formal language specifications. Static semantics is defined by an attribution technique (e.g. for the specification of flow graphs). The dynamic semantics is defined by ASMs. As an example, an object-oriented programming language with parallelism is specified. Part of this work has appeared as TR 93-1396 of Cornell University and in 1994 as *Developing Efficient Interpreters based on Formal Language Specifications* in P. Fritzson (Ed.): Compiler Construction, Springer LNCS 786, pages 233-247.

261. A. Prinz. *Formal Semantics for SDL. Definition and Implementation*. Habilitationsschrift, Humboldt University of Berlin, Germany, 2000.

    Contains a complete definition and implementation of the static and dynamic semantics of a characteristic sublanguage of SDL.

262. 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 [100].

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

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

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

    Systematic treatment of ASM models for Gödel [93], Parlog [92], Pandora [265], Concurrent Prolog [91], GHC. Thesis supervised by Börger.

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

    The ASM model for Parlog developed in [92] is extended by the don't-know nondeterminism of Pandora.

266. 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.

267. H. Rust. Hybrid Abstract State Machines: Using the Hyperreals for Describing Continuous Changes in a Discrete Notation. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines – ASM 2000, International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, number 87 in TIK-Report, pages 341–356. ETH Zürich, March 2000.

    A hybrid version of ASMs, incorporating the hyperreals for continuously changing quantities, is described.

268. H. Sasaki. A Formal Semantics for Verilog-VHDL Simulation Interoperability by Abstract State Machine. In *Proceedings of IEEE Conference DATE'99 (Design, Automation and Test in Europe), ICM Munich, Germany*, pages 353–357, March 9-12 1999.

    Based upon the VHDL models developed in [79, 80], a formal semantics for Verilog-HDL and VHDL focusing on the simulation model (with signal scheduling and time control) is defined. The semantics presented is faithful to the language reference manual and is proposed as first step towards semantic interoperability analysis on multi-semantic domains such as Verilog-AMS and VHDL-AMS. Extended in [270].

269. H. Sasaki. A New Dynamic Equation Scheduling to extend VHDL-AMS. In *Proceedings of Asia Pacific Conference on Chip Design Languages (APCHDL'99)*, pages 47–52, Fukuoka, Japan, 6-8 October 1999.

    An extension to VHDL-AMS for dynamic equation scheduling is proposed. The semantics of the extension is given in terms of the ASM model for VHDL-AMS presented in [271].

270. H. Sasaki. A Formal Semantics on Net Delay in Verilog-HDL. In *Proceedings of Asia Pacific Conference on Chip Design Languages (APCHDL'99)*, pages 100–106, Fukuoka, Japan, 6-8 October 1999.

    An extension of [268] giving semantics for net delays in Verilog-HDL using ASMs.

271. 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.

    An extension of the ASM model defined for VHDL in [79, 80] to provide a rigorous definition of VHDL-AMS, following the IEEE Language Reference Manual for the analog extension of VHDL. For an extension see [272]. See also [270, 269, 268].

272. T. Sasaki and H. Sasaki and K. Mizushima. Semantic Analysis of VHDL-ASM by Attribute Grammar. In *Proceedings of FDL'98 (Forum on Design Languages), Lausanne, Switzerland*, pages 123–131, September 6-10 1998.

    An extension of [271] to provide a formal semantics of the VHDL Analog Mixed Signal extension by means of attribute grammars. The formulation treats both static and dynamic aspects of semantics and permits one to show equality of process behavior.

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

    Published in: Dissertationen zur Künstlichen Intelligenz, vol.37, Infix-Verlag, Dr. Ekkehardt Hundt, St. Augustin 1993. Uses ASMs to define the semantics for the HERA language (and its implementation in Prolog), a special purpose programming language for the representation and manipulation of scheduling knowledge on the basis of heuristics, tailored to program efficient and reusable scheduling algorithms for production planning and control. See also J. Sauer, "Evolving Algebras for the Description of a Meta-Scheduling System", in H. Kleine Büning, ed., *Workshop der GI-Fachgruppe Logik in der Informatik*, Technical Report TR-RI-94-146, Universität Paderborn, 1994.

274. G. Schellhorn. *Verifikation abstrakter Zustandsmaschinen*. PhD thesis, Universität Ulm, 1999.

    ASMs are embedded into dynamic logic. Two refinement notions are extracted from typical ASM refinements and formalized in dynamic logic. A general modularisation theorem is proved for schemes to prove the correctness of refinements. An improved version of this theorem appears in [275]. The KIV system is enhanced to apply those proof techniques for a KIV verification of the WAM correctness proof in [100]. An english version of the thesis is available at Schellhorn's website.

275. G. Schellhorn. Verification of ASM Refinements Using Generalized Forward Simulation. *Journal of Universal Computer Science*, 7(11):952–979, 2001.

    See [274].

276. 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 Karlsruhe Interactive Verifier (KIV system) is applied to mechanically verify the proof of correctness of the Prolog to WAM transformation described in [100]. Starting point was the Diplom Thesis *Von Prolog zur WAM. Verifikation der Prozedurübersetzung mit KIV* of W. Ahrendt, Universität Karlsruhe (Germany) 1995. See comment to [100] and [274].

277. J. Schmid. Executing ASM Specifications with AsmGofer. Web pages http://www.tydo.de/AsmGofer.

The Web site for the machine to execute, equipped with graphical user interface, ASMs which are enhanced with the structuring and composition concepts defined in [103]. AsmGofer executes all the ASMs defined in [291]. In [116] AsmGofer has been used to build a simulator for UML state diagrams.

278. J. Schmid. Compiling Abstract State Machines to C++. *Journal of Universal Computer Science*, 7(11):1069–1088, 2001.

Introduces a scheme for compiling ASMs from the syntax of the ASM Workbench [123] to C++, coding algebraic types, pattern matching, functional expressions, dynamic functions, and simultaneous updates in such a way that efficient C++ code is obtained without loosing the structure of the original ASM specification. The compiler has been successfully applied in the FALKO project at Siemens [90]. In an early application C++ code is generated from a translation of the Production Cell ASM in [89] to the ASM Workbench format ASM-SL [123]. An HTML version is available at http://www.tydo.de/ProductionCell/.

279. J. Schmid. *Refinement and Implementation Techniques for Abstract State Machines*. PhD thesis, University of Ulm, Germany, 2002.

Thesis supervised by Börger and located at Siemens Corporate Research in München from August 1998 to July 2000. The thesis enriches ASMs by structuring and composition concepts [103] and their implementation in the AsmGofer system, developed for executing ASMs in an environment with graphical user interface. The concepts have been successfully applied in a middle sized software development project at Siemens [90, 278], in the Light Control Case Study [94], in an industrial ASIC design and verification project (including a compiler from ASM to VHDL), and for the modeling and implementation of Java and the JVM in [291].

280. P. Schmitt. Proving WAM Compiler Correctness. Technical Report 33/94, Universität Karlsruhe, Fakultät für Informatik, 1994.

Feasability analysis of Börger's proposal to the DFG project "Deduktion" to mechanize the Prolog-to-WAM compiler correctness proof in [100]. See [276, 275, 274, 262].

281. 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 KIV (Karlsruhe Interactive Verifier) to reason about ASMs directly. See [290].

282. W. Schönfeld. Interacting Abstract State Machines. In U. Glässer and P. Schmitt, editor, *Proceedings of the Fifth International Workshop on Abstract State Machines*, pages 22–36. Magdeburg University, 1998.

An extension to ASMs which permits one to specify forced synchronization of agent moves (à la Petri nets) is proposed and explored on some examples.

283. A. Schönhage. Storage Modification Machines. *Siam Journal of Computing*, 9:490–508, 1980.

Shown in [129] to be equivalent to a class of unary sequential ASMs.

284. 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 where multiple objects can establish cooperation contracts governing their answers to jointly received messages. An ASM rule is defined (Fig. 9 pg. 304) to formalize the run-time search of the most specific cooperation contract which implements a cooperative message. See [166].

285. I. Soloviev. *Exploration and experimental implementation of recursive patterns and functions imbedding into Prolog language syntactical environment.* PhD thesis, St. Petersburg University, Russia, 1995.

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

286. M. Spielmann. Automatic Verification of Abstract State Machines. In N. Halbwachs and D. Peled, editors, *Proceedings of 11th International Conference on Computer-Aided Verification (CAV '99)*, volume 1633 of *LNCS*, pages 431–442. Springer-Verlag, 1999.

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

287. M. Spielmann. *Abstract State Machines: Verification Problems and Complexity.* PhD thesis, University of Aachen, Germany, 21.6. 2000.

Investigation of the complexity of decision problems for certain classes of ASMs. Most of the results appear in [286, 288, 289]. The second part of the thesis relates to the work in [41]. A restricted ASM model to capture log-space computable functions on structures is defined, see also[169].

288. M. Spielmann. Verification of Relational Transducers for Electronic Commerce. In *Proceedings of 19th ACM Symposium on Principles of Database Systems (PODS 2000).* ACM Press, 2000.

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

289. M. Spielmann. Model Checking Abstract State Machines and Beyond. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 323–340. Springer-Verlag, 2000.

Decision problems for ASMs are investigated, i.e. problems to decide, for an ASM M of a given class and for a property P of a given form, whether M satisfies P. For particular classes of machines and of property describing formulae, the computational complexity of such problems is studied for the following two cases: a) given M, P, and input I, decide whether P holds during all M-computations over I (called model-checking problem); b) given M, P, decide whether for every input I, P holds during all M-computations over I (called verification problem). Appeared also as TIK-Report 87, ETH Zürich, March 2000, 357–375.

290. R. Stärk and S. Nanchen. A Logic for Abstract State Machines. *Journal of Universal Computer Science*, 7(11):981–1006, 2001.

A new logic for sequential, non-distributed ASMs is presented which is based on an atomic predicate for function updates and on a definedness predicate for the termination of the evaluation of ASM rules. The logic allows for sequential and hierarchical recursive submachine composition as defined in [103]. It is proven complete for hierarchical non-recursive ASMs. This logic provides a unifying view of the logics for ASMs developed in [172, 281, 259, 147]. A preliminary version appeared in L. Fribourg (Ed.): Computer Science Logic (CSL 2001), Springer LNCS 2142, pp. 217-231, 2001.

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

A high-level description, together with a mathematical and an experimental analysis (verification and validation), of Java and of the Java Virtual Machine (JVM), including a standard compiler of Java programs to JVM code and the security critical bytecode verifier component of the JVM. Includes an executable ASM specification written for AsmGofer. For an evaluation see [198, Section 6.2], for more information see `http://www.inf.ethz.ch/~jbook/`.

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

293. Kurt Stenzel. Verification of JavaCard Programs. Technical report 2001-5, Institut für Informatik, Universität Augsburg, Germany, 2001. Available at http://www.Informatik.Uni-Augsburg.DE/swt/fmg/papers/. The report is about the formal verification of JavaCard or sequential Java programs (i.e. without synchronized statements). A calculus in dynamic logic is defined and implemented in KIV. KIV parses the original JavaCard or Java program, resolves names and types in the same manner as a normal Java compiler, and produces an annotated abstract syntax tree that is the input for the verification. All sequential Java statements are supported, including exceptions, breaks, static initialization, objects, dynamic method lookup, and arrays. The abstract syntax of Java programs, the proof rules, and the underlying algebraic specifications for the object store and the primitive data types, and a formal semantic is described in detail. An example proof and a list of validation programs conclude the report. For information on preliminary work on formalizing ASM models for Java in KIV see http://www.informatik.uni-augsburg.de/swt/fmg/applications/ and http://www.informatik.uni-augsburg.de/swt/fmg/applications/javaASM.html.

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

Upon Börger's suggestion, 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.

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

Develops a language for specifying software systems by linking components via connectors. Components are abstractly characterized by the services they import and export which are defined by high-level specifications (possibly depending on given views) and have to satisfy certain constraints on well-formedness and on the ordering of usage (called use structure). For connectors, which connect services required in one component to services offered by other components, a refinement concept is defined. ASM rules are provided to check the consistency of software architectures developed in that language, namely checking componentwise a) for each imported service its correct connection to a corresponding exported service (wrt signature and specification), b) for each exported service that the imported services it uses satisfy the constraints of the used components, c) that the (optional) refinement is correct wrt the system constituents (types, views, components, connectors). The proposed machine has been made executable in XASM. Extended abstracts of some of the ideas in the thesis have been published by M. Anlauff and A. Sünbül as *Software Architecture Based Composition of Components* (Gesellschaft für Informatik, Sicherheit und Zuverlässigkeit software-basierter Systeme, May 1999), *Component Based Software Engineering for Telecommunication Software* (Proc. SCI/ISAS Conf., Orlando, Florida 1999), *Domain-Specific Languages in Software Architecture* (Proc. of the Integrated Design and Process Technology IDPT99, June 1999).

296. J. Teich. Project buildabong at university of paderborn. http://www-date.upb.de/RESEARCH/BUILDABONG/buildabong.html, 2001.

The project, led by Teich at the University of Paderborn, uses ASMs to provide behavioral and structural descriptions of application-specific instruction set processors, from which (using XASM [8] and Gem-Mex [10]) bit-true and cycle-accurate simulators and debuggers are derived. See the paper "Design Space Characterization for Architecture/Compiler Co-Exploration" by D. Fischer, J. Teich, R. Weper, U. Kastens, M. Thies in: Proceedings of ACM Conference CASES'01, November 16-17, 2001, Atlanta, Georgia, USA.

297. J. Teich, P. Kutter, and R. Weper. Description and Simulation of Microprocessor Instruction Sets Using ASMs. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 266–286. Springer-Verlag, 2000.

A method for transforming register transfer descriptions of microprocessors into executable ASM specifications is described and illustrated using the ASM model developed in [206] for the ARM2 RISC processor. The description exploits the natural correspondence between the simultaneous execution of all guarded update rules of an ASM and a single-clock hardware step executing a set of Leuper's guarded register transfer patterns. XASM [8] is used together with the Gem-Mex tool [10] which generates a graphical simulator for the given architecture. See also [298]. Also appears in TIK-Report 87, ETH Zürich, March 2000, 376–397.

298. J. Teich, R. Weper, D. Fischer, and S. Trinkert. A Joint Architecture/Compiler Design Environment for ASIPs. In *Proc. International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES2000)*, pages 26–33. ACM, November 2000.

An ASM model is developed for a VLIW digital signal processor of the Texas Instruments TMS320 C6200 family to illustrate the Buildabong method [296]. See also [297].

299. 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. A first part appeared under the title "A Formalization of Many-sorted Evolving Algebras" as TR 93-115 at TU Delft. An extended abstract appeared under the title *A Sound and Complete SOS-Semantics for Non-Distributed Deterministic Abstract State Machines* in [161, 91-110].

300. 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.

301. M. Vale. The Evolving Algebra Semantics of COBOL. Part I: Programs and Control. Technical Report CSE-TR-162-93, EECS Dept., University of Michigan, 1993.

An ASM for the control constructs of COBOL. A description of a plan for a series of ASMs for all of COBOL is sketched (but not carried out). Missing constructs concern source text manipulations, report writer, communication, debug, segmentation modules.

302. 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.

303. 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 description in [184] of the semantics of C is extended to C++.

304. C. Wallace. The Semantics of the Java Programming Language: Preliminary Version. Technical Report CSE-TR-355-97, EECS Dept., University of Michigan, December 1997.

A specification of the static and dynamic semantics of Java, using ASMs and Montages. This work showed the shortcomings of the original formulation of Montages [224] and led to its state machine based reformulation in [11]. See [223] and the independent earlier Java modeling work [106] which was continued in [107, 108, 109, 110] and [291].

305. C. Wallace, G. Tremblay, and J. N. Amaral. An Abstract State Machine Specification and Verification of the Location Consistency Memory Model and Cache Protocol. *Journal of Universal Computer Science*, 7(11):1089–1113, 2001.

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

Inspired by Börger's lectures on ASMs in Freiburg in the Fall of 1994, Winter develops a framework for using a model checker to verify ASM specifications. It is applied to the production cell control model described in [89]. See [256] for an interesting problem with refining abstractions for model checking purposes. For an extension see [128, 307, 309, 310].

307. K. Winter. Towards a Methodology for Model Checking ASM: Lessons Learned from the FLASH Case Study. In Y. Gurevich and P. Kutter and M. Odersky and L. Thiele, editor, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 341–360. Springer-Verlag, 2000.

A general discussion of applying model checking to ASMs. Following a suggestion by Börger, the ASM specification of the FLASH cache coherence protocol [137] is checked using SMV as a case study. An extension of [128, 306]. Also appears in TIK-Report 87, ETH Zürich, March 2000, 398–425.

308. K. Winter. Automated Checking of Control Tables. E-mail to E. Börger, December 24, 2001.

Case study for automated checking of Control Tables, used by the Software Verification Research Centre at the University of Queensland, Australia, to specify railway interlocking systems. The control tables are formalized as ASMs and then transformed by the algorithm described in [128] to become input for the SMV model checker.

309. K. Winter. *Model Checking Abstract State Machines.* Ph.d. thesis, Technical University of Berlin, 2001.

Based upon [306, 128, 307, 310], a transformation of ASMs to FSMs and abstraction mechanisms in the context of model checking large ASMs are investigated and implemented. The underlying tools are the ASM Workbench [123], SMV and Multiway Decision Graphs (for the latter see also [310]).

310. K. Winter. Model Checking with Abstract Types. In S. D. Stoller and W. Visser, editors, *Workshop on Software Model Checking*, volume 55 (3) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V., Paris (France) July 23 2001.

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

311. A. Zamulin. Typed Gurevich Machines Revisited. Joint CS & IIS Bulletin, Computer Science 7 (95-122), 1997.

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

312. 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 [311] is used to define a compiler for Oberon to an algebraically-specified abstract target machine.

313. 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.

314. A. Zamulin. Object-Oriented Abstract State Machines. In U. Glässer and P. Schmitt, editors, *Proceedings of the Fifth International Workshop on Abstract State Machines*, pages 1–21. Otto-von-Guericke-Universität Magedeburg, 1998.

Proposes an extension of ASMs to include objects.

315. A. Zamulin. Specification of Dynamic Systems by Typed Gurevich Machines. In Z. Bubnicki and A. Grzech, editors, *Proceedings of the 13th International Conference on System Science*, pages 160–167, Wroclaw, Poland, 15-18 September 1998.

A combination of many-sorted algebraic specifications for states and ASM-rules for transitions is proposed as an approach for dynamic system specification. The approach is used in [312] to specify an Oberon compiler.

316. A. Zamulin. Generic Facilities in Object-Oriented ASMs. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines – ASM 2000, International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, number 87 in TIK-Report, pages 426–446. ETH Zürich, March 2000.

The object-oriented ASM framework introduced in [314] is extended to allow the definition of generic object types, type categories, functions, and procedures. Examples from the C++ Standard Template Library (STL) are provided. Previously appeared in Preprint 60, Institute of Informatics Systems, Siberian Division of the Russian Academy of Sciences, Novosibirsk, 1999.

317. A. Zamulin. Specifications in-the-Large by Typed ASMs. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines – ASM 2000, International Workshop on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings*, number 87 in TIK-Report, pages 447–461. ETH Zürich, March 2000.

A discussion of combining typed ASMs (as proposed in [315]) to produce larger ASMs.

318. 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).

319. 1996 ASM Community. Name Change to Replace EA by Something Better. Electronic Discussion at ea@ira.uka.de, September 6 to October 11 (1996).

The discussion was proposed with the following motivation: ""Algebra" makes the theoreticians think that the approach belongs to the algebraic specification and verification research area - and their dissatisfaction and misjudgement comes from our violating so many (I would say almost all) of their cherished concepts and beliefs. "Algebra" makes the practitioners think that we want them to use algebraic notation and equations or laws - and this is enough for them not even to look further what we really do. "Evolving" is either not understood at all or in the best of all cases interpreted as implying that the signature should be allowed to change - this comes from the analogy with biological systems where the concept is used that way." (Börger on Sept 6)
In a lively discussion, two dozens of names were proposed, resulting in Päppinghaus' proposal (Gurevich's) *Abstract State Machines* to become generally accepted. Here are the concluding messages of October 10/11 which resume this decision.
From: Erik Tiden Erik.Tiden@zfe.siemens.de To: eboerger@prosun.first.gmd.de Subject: Name of the beast.
Dear Prof. Boerger, I write in English, so that you can quote me to your community if you wish. The name "Gurevich Machines" is impossible in industry, because it only evokes associations of useless (in industrial practice) theoretical concepts. The name "Abstract State Machines" on the other hand, is fine. That's also what we will keep on calling them here at Siemens central research. Thus, if you stick to "Gurevich Machines" you will end up with two names. Now, if you regard ASMs as a theoretical excercise, investigation, whatever, into the foundations of CS or some such worthy cause, then you can call them whatever you like of cause. If you want to make ASMs into something which is useful in practice, calling them GM is simply foolish. Best regards, Erik Tiden.
Answer of October 11. From: Egon Boerger eboerger@prosun.first.gmd.de To: Erik.Tiden@zfe.siemens.de Subject: Abstract State Machines (Gurevich Machines).
Dear Dr. Tiden, thanks for your valuable comment on the EA name problem which I am going to answer in English so that the whole community can follow this. I do not know whether you did follow the entire discussion; I had started it pushed by the need to find a name which helps those of us who aim at practical (in particular industrial) applications of the specification, verification and code development method which has been built around Gurevich's notion of evolving algebras. I am glad that through the discussion we have found such a name, namely Gurevich ('s Abstract State) Machines. By the way, the first step to this solution, namely the proposal to call the beasts Abstract State Machines, came from one of your collaborators, Dr. Paeppinghaus, to whom I am grateful for his

suggestion. Adding the inventor's name to Abstract State Machines is in accordance with usual practice and provided the chance to conclude the search not with two really different names (EA and ASM) but with ONE name which is generally accepted by the community. Gurevich Machines or Abstract State Machines are not two different names but only shorthands for Gurevich's Abstract State Machines. Here is another variation, appearing in the title for one of my forthcoming lectures: Abstract State Machines (Gurevich Machines). An interesting feature which makes Gurevich's ASMs unique is that they have both practical AND theoretical relevance (although surprisingly enough the theoretical potential of the notion of Gurevich Machines has been recognized and explored even less than its practical relevance). Therefore it IS valuable to have a unique name which takes into account the sometimes diverging interests. I hope this is a satisfactory answer to your message. With best wishes, Egon Boerger.