Journal of Universal Computer Science, vol. 3, no. 5 (1997), 414-415
submitted: 20/12/96, accepted: 20/5/97, appeared: 28/5/97 © Springer Pub. Co.

JUCS Special ASM Issue. Part II.

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

In JUCS 3.4 we have explained the context of "Ten Years of Gurevich's Abstract State Machines" of the special ASM issue for which Part II is appearing now. As announced at the end of Part I, this second part is devoted to applications of ASMs to classical problems of programming and software engineering. Two papers deal with semantics of programming languages, two papers with methods for building correct compilers and three papers with integrating ASMs into the software development life cycle.

Before I summarize the papers appearing here let me provide some technical data on this special ASM issue of J.UCS. An international call for submission of papers has been distributed in the first half of 1996, the deadline was December 1996. The first round of the reviewing process was finished at the beginning of March, the second round at the beginning of May 1997. Assisted by 76 review reports (ideally four per submission) I have accepted 7 papers for the April issue and 7 for the May issue, out of 21 submissions. One paper is still in the reviewing process and may appear in the next issue of J.UCS.

In the paper *Montages Specifications of Realistic Programming Languages* by P. Kutter and A. Pierantonio the successful application of ASMs to the description of the dynamics of real programming languages—like Prolog, C, Occam, VHDL, C^{++}—is enhanced by a semi-visual formalism that allows one to integrate into the ASM specification method a transparent formalization of the relevant static programming language features. In the paper *The Formal Specification of Oberon* the same authors illustrate their method through a formal but relatively compact ASM specification of the semantics of Oberon which includes both the dynamic and the static aspects (including the syntax).

The paper *On the Construction of Correct Compiler Back-Ends: An ASM Approach* by W. Zimmermann and T. Gaul uses ASMs for provably correct bottom-up rewriting systems specifications for back-end compilers which translate intermediate languages (basic block graphs) into binary machine code of a register based RISC processor with performance of the same order of magnitude as code generated by non-optimizing unverified C-compilers. The proofs are general in the sense that they make no specific assumptions on the instruction set of the intermediate language and of the target machine so that really a generator is defined which is parameterized by a term rewrite system, the intermediate and the target language and a register assignement algorithm. It is remarkable that all proofs in this paper have been machine checked using PVS, thus showing in particular that PVS can be put to use in a fruitful way to machine check ASM based reasoning on program development. The paper *Correctness Proof of a Distributed Implementation of Prolog by Means of ASMs* by L. Araujo specifies an extension of the WAM for (AND- and OR-) parallel execution of Prolog on distributed memory and proves its correctness. The construction illustrates nicely the rather typical reusability property of ASM specifications and verifications: it builds upon and extends the specification and correctness proof of the sequential WAM for ISO standard Prolog developed in [Börger and Rosenzweig 1994].

The three last papers deal with *Integrating ASMs into the Software Development Life Cycle* as the title of the paper by myself and L. Mearelli says. This paper presents a structured software engineering method which allows the software engineer to control efficiently the modular development and the maintenance of well documented, formally inspectable and smoothly modifiable code out of rigorous ASM models for requirement specifications. It defines an ASM model for the requirement specifciation and refines it (in a way which is proved to be correct) to a model which in the paper *Refining an ASM Specification of the Production Cell to C++ Code* by L. Mearelli is then refined to C^{++} code which has been validated through extensive experimentation. The paper also 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 have been computer verified by a model checking approach for ASMs developed by K. Winter and illustrated in her paper *Model Checking for Abstract State Machines.*

We hope the reader will benefit from the ASM papers appearing here and will be tempted to try the method for his next challenging application problem.

References

[Börger and Rosenzweig 1994] E. Börger and D. Rosenzweig: "The WAM Definition and Compiler Correctness"; C.Beierle, L.Plümer (Eds.), Logic Programming: Formal Methods and Practical Applications, North-Holland, Series in Computer Science and Artificial Intelligence (1994), 20-90