A Tribute to Dean Rosenzweig

Egon Börger

Università di Pisa, Dipartimento di Informatica, I-56125 Pisa, Italy boerger@di.unipi.it

1 Dubrovnik

Dean Rosenzweig passed away in the second week of January 2007. I met Dean for the first time 24 years ago, during the winter School *Foundation of Computation Theory* Profs. Rasiowa, Karpinski and Kirin had organized at the Inter-University Centre for Post-graduate studies in Dubrovnik from 16.1. - 29.1.1983. Dean attended my lectures on *Complexity of Logical Theories* and I noticed him as a particularly bright, sharp and very knowledgeable young colleague with a strong taste for elegance in mathematical matters and a wide range of interests. I vividly remember strolling with Dean and his friends long night hours through the streets of the historic Dubrovnik and along the town wall, discussing issues of logic, complexity, semantics whilst enjoying the splendid view on the town, its harbor, the hills and the sea. We shared the love for Dubrovnik and the town became our fate.

2 ASM Models for Logic Programming Concepts

In fact we met again in Dubrovnik seven years later, during two invited lectures I delivered at the *Logic and Computer Science* Conference (LIRA, 6.9. - 9.9.1990) and the *International Summer Seminar on Artificial Intelligence* (CAS, 3.9. - 7.9.1990). Dean was again in the audience when I explained *The Dynamic Algebra Approach to Semantics of Prolog and Prolog III* and the use of ASMs to construct a practically useful *Formal Model for Semantics of Constraint Logic Programming Systems*. In those lectures I presented the use of ASMs to build *ground models* [10] for logic programming systems and explained among others the problem Prolog experts had challenged me to solve as test for the usefulness of such models, namely to explain through them the intricate optimization features of the Warren Abstract Machine implementation of Prolog and to verify the correctness of the WAM, which represented the state-of-the-art of logic programming practice and at the time was understood only by a small number of experts worldwide.

Dean enthousiastically joined the project to which he brought his detailed knowledge of logic programming implementation issues.[1] Through intensive work both of us enjoyed enormously, started right after my talks in Dubrovnik and going through the fall and winter of 1990/91, when Dean visited me repeatedly at IBM in Heidelberg where I was spending a sabbatical and in Pisa, we built a series of 12 stepwise refined ASM models linking my ASM model of Prolog [4,5] to an ASM model for the Prolog implementation by WAM code, proving the correctness of each refinement step and thus verifying the WAM implementation of Prolog. Less than a year later I could report the successful outcome of this work in an invited talk on the *Correctness proof for a class of Prolog Compilers on Warren's Abstract Machine*, delivered to the 13th International Conference on *Information Technology Interface* (ITI'91) in Dubrovnik-Cavtat (10.6. - 14.6.1991), with Dean again present in the audience, and published in [25,26,30].

The Prolog-to-WAM work represented the first full-fledged ASM method case study that proved the potential of using ASMs for a practical and effectively verifiable design of a complex system, from its high-level specification to an optimized implementation. For three reasons the work became a landmark in the short history of ASMs.

[1] On Dean's web page I found a reference to an apparently unpublished paper [44] he had never mentioned to me that seems to document the source of this knowledge.

- The WAM paper represents the first extensive field test for the *practicality of the ASM refinement concept* [11], which became one of the three constituents of the Asm method [12], together with the concept of ASM ground models [10] and the very notion of ASM [37], and had been defined and adopted for the first time in [4,5,7] to construct the Prolog ground model, starting from a four-rule kernel for core Prolog and extending it by specific machines for the various types of built-in predicates. The Prolog-to-WAM refinement hierarchy extended by vertical refinements the horizontal refinements that had been used for building the Prolog ground model. Whereas the horizontal refinements support a separation of orthogonal language features by modules of rule sets and have been reused for numerous adaptations and extensions of Prolog (surveyed in [9] and [32, Sect.9.2]) and later for richer languages (e.g. Java [55] and C# [19]), the vertical refinement steps add the definition and analysis of those features at successively richer levels of implementation detail. In fact for the KIV verification effort of Prolog-to-WAM (see the next bullet), ASMs had to be embedded into dynamic logic together with a formalization of two specializations of the ASM refinement notion, for which Schellhorn [49] proved a general modularisation theorem and used it as scheme to prove the correctness of the ASM refinement steps in the Prolog-to-WAM hierachy. An improved version of this theorem appeared later in [50], see also the continuation of the analysis of the ASM refinement notion in [51].
- The WAM paper triggered the first full-fledged *machine-supported ASM verification effort*. My suggestion to the German Research Council (DFG) project "Deduktion" to mechanize the Prolog-to-WAM compiler correctness proof was taken up by two theorem proving groups, resulting in an attempt to use Isabelle [41] for this purpose and in a complete mechanical KIV verification of the entire refinement chain [54,52,53,49].
- The WAM paper has triggered the first illustrations of *effective reuse of ASMs*, namely by adapting the Prolog-to-WAM ASM refinement hierarchy and the related correctness proofs for other logic programming languages and their proven to be correct implementations, e.g. of CLP($\mathcal{R}$) on IBM's constraint logical arithmetical machine [31], of PROTOS-L on IBM's WAM extension for type-constraint logic programming [3,2] and of the Prolog Distributed Processor extending the WAM for parallel execution of Prolog on distributed memory [1]. Through this work we learnt much of what later gave me the confidence to embark on an ASM analysis of Java and C# and their implementation on the Java Virtual Machine respectively the .NET Common Language Runtime [55,19,33,34].

At this point, having experienced through the Prolog-to-WAM work the huge potential the ASM method offers for mathematically rigorous and elegant system design and verification work, Dean was hooked to the method and became an affecionado for the rest of his life. Since he was one of the not too numerous persons who combine a strong love for pure mathematics with the readiness to apply his mathematical skills to address practical problems, it was not any more difficult to convince him to join me also for the collaboration I had started with the ISO Prolog standardization committee [14]. If the ASM approach to defining the semantics of Prolog for a long time encountered much incomprehension and considerable resistance in the logic programming and language semantics communities, this has not been the case within the ISO standardization committee, where the partners were language designers and software engineers responsible for the at the time leading logic programming systems from Quintus, IBM, Siemens, Interface and BIM. Already in 1991, in a second invited talk to the ITI'91 Dubrovnik-Cavtat conference, I could present the work with Dean on *An Analysis of Database Views and their Uniform Implementation*, which had been triggered by the discussions in the standardization group [16][2]. In the paper [23] with Dean we used the ASM model for Prolog to solve the *Problems with assert, retract and abolish in Prolog*, theme of a talk we presented together in the same summer to the ISO WG 17 Meeting in Paris (01.07. - 03.07.1991). Also the paper [28] belongs here, where we provided an ASM-based specification of the solution-collecting predicates *findall, setof,bagof* of Prolog, which paved the way for a logico–mathematical analysis, rationale and criticism of various proposals

[2] See the preliminary version *The View on Database Updates in Standard Prolog: A Proposal and a Rationale* published as ISO/ETC JTCI SC22 WG17 Prolog Standardization Report no. 74, February 1991, pp. 3–10.

made for implementations of these predicates. The paper was also issued as *Prolog. Copenhagen Papers 2*, ISO/IEC JTC1 SC22 WG17 Standardization report no. 105, by the National Physical Laboratory in Middlesex in 1993 (pp. 33–42).

As a sequel to the Prolog-to-WAM work we streamlined the early Prolog models [4,5,7,8,6], work that was reported by Dean in form of a poster presentation *Full Prolog in a Nutshell* at the *10th International Conference on Logic Programming* (ICLP '93) in Budapest (21.06. - 24.06.1993) and published in [24,27,29].

As was typical for Dean, he would try to extract from the concrete work done some theoretical insight. In [36] he defined an ASM interpretation of many-step SOS, denotational semantics and Hoare logic for the language of while-programs. Correctness and completeness theorems are stated based on a simple flowchart model of the language.

3 ASM Models of Concurrency Concepts

When in the Fall of 1992 I started the project to systematically test the practical impact of the ASM method beyond the semantics of programming languages, by trying out ASMs for the modeling and rigorous mathematical and experimental analysis of a variety of complex real-life computer-based systems, one of the first relevant fields I investigated was computer architecture. Dean joined me, again enthousiastically, in the reverse engineering study commissioned by a group of physicists in Pisa and Rome for the massively parallel APE100 architecture. As starting point for the work in Giuseppe Del Castillo's Tesi di Laurea dedicated to this study, we developed with Dean and Paola Glavan, at the time one of Dean's students, a programmer's view ground model [13] for this architecture, which was presented by Dean in his talk on *A formal model for the APE100 architecture viewed through the APESE language* at the Physics Department of Università di Pisa on 28.10.1993 and then refined in [15] 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.

With Igor Durdanovic, the second student Dean brought into the ASM community, we developed a truly concurrent ASM ground model for Occam, which I presented in a talk to the Procomet'94 IFIP Working Conference on Programming Concepts, Methods and Calculi and which was published in the Proceedings [18]. The model improved the parse tree determined ASM model in [38]. It has been enriched in [17] by a series of refinement steps to a proven to be correct Transputer implementation of Occam.

This work inspired Dean to develop within the framework of ASMs a theory of concurrent computation, which was illustrated in [35] (see also [42]) by ASM models for the Chemical Abstract Machine and the π-calculus and generalized the approaches developed in [38,21,22] in connection with ASM models for Occam, Concurrent Prolog and Parlog. It was superseded only by the more general definition of distributed ASM runs in the Lipari Guide [37]. Using the latter notion, with Yuri Gurevich we analyzed in [20] Lamport's Bakery algorithm—first of all in terms of an ASM ground model to faithfully reflect Lamport's mutual exclusion protocol. By abstracting from the low-level read and write operations of the ground model, we defined a high-level model with atomic actions (non-overlapping reads and writes) for a simple proof of the desired correctness and liveness properties from simple axioms. In a third refinement step we turned atomic into durative actions, allowing overlapping of reads and writes to shared registers, and proved that the corresponding assumptions made for the machine with atomic actions still hold. This work has later been analyzed again by Dean and Gurevich [39] in terms of partially ordered runs of distributed ASMs, abstracting from the mapping of moves to linear real time we had used in our original proofs in [20].

During this experimentation period, which was important for the development of the ASM method through its applications, Dean also helped me to eventually convince Yuri to dedicate the special session, which Gurevich had been asked to organize as part of the IFIP 13th World Computer Congress in Hamburg, not to complexity theory but to ASMs, which at the time were called Evolving Algebras. The meeting we helped Yuri to organize (see the chapter on Stream C on Evolving Algebras in the Proceedings [40, 377-441]) became a rather successful start of the series of annual international ASM workshops, the 2^4th one of which Dean has been planning to hold in 2009 in Dubrovnik, the town where then two decades ago he had been attracted to ASMs. I was dreaming about iterating my walks with Dean through the enchanted place...

4 Conclusion

Dean's wide spread interest also covered logic and recently cryptographic protocols. He related ASMs to linear logic [43], studied typed hybrid multimodal logic [45] and exploited ASMs for poblems in cryptography [48,46,47]. These interests are also reflected in the subjects Dean lectured about. The graduate courses he taught over the last six years were on cryptographic protocols, logics of knowledge and belief for cryptographic protocol analysis, applied logic, higher order logic, hard algorithmic problems and a programming course *Processing of Natural Language*.

With Dean a part of my own research life has gone, related to the crucial period in the first half of the 90'ies of the last century when through a collaborative effort involving dozens of colleagues on the two sides of the Atlantic we slowly learnt enough about complex real-life computer-based systems and their implementations to eventually understand and formulate the ASM method as an accurate, scientifically well-founded *engineering method* that supports the trustworthy design, analysis and documentation of such systems also under industrial constraints. Dean's insightful contributions to this effort will not be forgotten by all those who had the chance to work with him.

References

1. L. Araujo. Correctness proof of a distributed implementation of Prolog by means of Abstract State Machines. *J. Universal Computer Science*, 3(5):416–422, 1997.
2. 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.
3. 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.
4. 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 *Lecture Notes in Computer Science*, pages 36–64. Springer-Verlag, 1990.
5. 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 *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 1990.
6. E. Börger. Dynamische Algebren und Semantik von Prolog. In E. Börger, editor, *Berechenbarkeit, Komplexität, Logik*, pages 476–499. Vieweg, 3rd edition, 1992.
7. E. Börger. A logical operational semantics for full Prolog. Part III: Built-in predicates for files, terms, arithmetic and input-output. In Y. N. Moschovakis, editor, *Logic From Computer Science*, volume 21 of *Berkeley Mathematical Sciences Research Institute Publications*, pages 17–50. Springer-Verlag, 1992.
8. E. Börger. A natural formalization of full Prolog. *Newsletter of the Association for Logic Programming*, 5(1):8–9, 1992.
9. 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, 1994.
10. E. Börger. The ASM ground model method as a foundation of requirements engineering. In N.Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 145–160. Springer-Verlag, 2003.
11. E. Börger. The ASM refinement method. *Formal Aspects of Computing*, 15:237–257, 2003.
12. E. Börger. Construction and analysis of ground models and their refinements as a foundation for validating computer based systems. *Formal Aspects of Computing*, 2006.
13. E. Börger, G. D. Castillo, P. Glavan, and D. Rosenzweig. Towards a mathematical specification of the APE100 architecture: the APESE model. In B. Pehrson and I. Simon, editors, *IFIP 13th World Computer Congress*, volume I: Technology/Foundations, pages 396–401, Elsevier, Amsterdam, 1994.
14. 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.

15. 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, *Proc. 1st IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS'95)*, pages 145–148, November 1995.
16. E. Börger and B. Demoen. A framework to specify database update views for Prolog. In M. J. Maluszynski, editor, *PLILP'91. Third Int. Sympos. on Programming Languages Implementation and Logic Programming.*, volume 528 of *Lecture Notes in Computer Science*, pages 147–158. Springer-Verlag, 1991.
17. E. Börger and I. Durdanović. Correctness of compiling Occam to Transputer code. *Computer Journal*, 39(1):52–92, 1996.
18. E. Börger, I. Durdanović, and D. Rosenzweig. Occam: Specification and compiler correctness. Part I: Simple mathematical interpreters. In U. Montanari and E. R. Olderog, editors, *Proc. PROCOMET'94 (IFIP Working Conf. on Programming Concepts, Methods and Calculi)*, pages 489–508. North-Holland, 1994.
19. E. Börger, G. Fruja, V. Gervasi, and R. Stärk. A high-level modular definition of the semantics of C#. *Theoretical Computer Science*, 336(2–3):235–284, 2005.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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 *Lecture Notes in Computer Science*, pages 31–66. Springer-Verlag, 1991.
26. 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-Verlag, 1992.
27. E. Börger and D. Rosenzweig. Full prolog in a nutshell. In F.D.S.Warren, editor, *Logic Programming*, page 832. MIT Press, 1993.
28. 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 *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1993.
29. E. Börger and D. Rosenzweig. A mathematical definition of full Prolog. *Science of Computer Programming*, 24:249–286, 1995.
30. 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*, volume 11 of *Studies in Computer Science and Artificial Intelligence*, chapter 2, pages 20–90. North-Holland, 1995.
31. 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.
32. E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis.* Springer, 2003.
33. N. G. Fruja. A modular design for the Common Language Runtime (CLR) architecture. In D. Beauquier, E. Börger, and A. Slissenko, editors, *Proc.ASM05*, pages 175–200. Université de Paris 12, 2005.
34. N. G. Fruja and E. Börger. Modeling the .NET CLR Exception Handling Mechanism for a Mathematical Analysis. *Journal of Object Technology*, 5(3):5–34, 2006. .
35. 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-Verlag, 1993.

36. 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, 1994.
37. Y. Gurevich. Evolving algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
38. Y. Gurevich and L. S. 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 *Lecture Notes in Computer Science*, pages 176–192. Springer-Verlag, 1990.
39. Y. Gurevich and D. Rosenzweig. Partially ordered runs: A case study. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines: Theory and Applications*, volume 1912 of *Lecture Notes in Computer Science*, pages 131–150. Springer-Verlag, 2000.
40. B. Pehrson and I. Simon. I: Technology/foundations. In *IFIP 13th World Computer Congress 94*, Elsevier, Amsterdam, 1994.
41. C. Pusch. Verification of compiler correctness for the WAM. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics (TPHOLs'96)*, volume 1125 of *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, 1996.
42. 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, 1994.
43. D. Rosenzweig. Evolving algebras and light linear logic. *Bulletin of Symbolic Logic*, 3, 1997.
44. D. Rosenzweig, I. Durdanovic, and I. S. et al. A new prolog compiler - zglog. In *Computer Aided Systems Conference (CAS'90)*, Dubrovnik, 1990.
45. D. Rosenzweig and D. Runje. Tableaux-based prover for typed hybrid multimodal logic (system description). In *3rd Method for Modalities Workshop*. INRIA-Lorraine (Nancy), 2003.
46. D. Rosenzweig and D. Runje. The cryptographic abstract machine. In *Abstract State Machines–Advances in Theory and Applications*, volume 3065 of *Lecture Notes in Computer Science*, pages 202–217. Springer-Verlag, 2004.
47. D. Rosenzweig, D. Runje, and W. Schulte. Model-based testing of cryptographic protocols. In *Trustworthy Global Computing: IST/FET International Workshop*, pages 33–60, 2005.
48. D. Rosenzweig, D. Runje, and N. Slani. Privacy, abstract encryption and protocols: an ASM model–part I. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Abstract State Machines 2003–Advances in Theory and Applications*, volume 2589 of *Lecture Notes in Computer Science*, pages 372–390. Springer-Verlag, 2003.
49. G. Schellhorn. *Verifikation abstrakter Zustandsmaschinen*. PhD thesis, Universität Ulm, Germany, 1999.
50. G. Schellhorn. Verification of ASM refinements using generalized forward simulation. *J. Universal Computer Science*, 7(11):952–979, 2001.
51. G. Schellhorn. ASM refinement and generalizations of forward simulation in data refinement: A comparison. *Theoretical Computer Science*, 336(2-3):403–436, 2005.
52. G. Schellhorn and W. Ahrendt. Reasoning about Abstract State Machines: The WAM case study. *J. Universal Computer Science*, 3(4):377–413, 1997.
53. G. Schellhorn and W. Ahrendt. The WAM case study: Verifying compiler correctness for Prolog with KIV. In W. Bibel and P. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume III: Applications, pages 165–194. Kluwer Academic Publishers, 1998.
54. P. Schmitt. Proving WAM compiler correctness. Technical Report 33/94, Universität Karlsruhe, Fakultät für Informatik, Germany, 1994.
55. R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation*. Springer-Verlag, 2001. .