1 Introduction

In 1983, David H. D. Warren designed an abstract machine for the execution of Prolog consisting of a memory architecture and an instruction set [War83]. This design became known as the Warren Abstract Machine (WAM) and has become the *de facto* standard for implementing Prolog compilers. In [War83], Warren describes the WAM in a minimalist's style, making understanding very difficult for the average reader, even with a foreknowledge of Prolog's operations. Too much is left untold, and very little is justified in clear terms.[1] This has resulted in a very scant number of WAM *aficionados* who could boast understanding the details of its workings. Typically, these have been Prolog implementors who decided to invest the necessary time to learn by doing and to reach enlightenment painstakingly.

1.1 Existing literature

Witness to this lack of understanding is the fact that in six years there has been little published that would teach the WAM, let alone formally justify its correctness. Indeed, besides Warren's original hermetic report [War83], there has been virtually no official publication on the WAM. A few years ago, one could come across a draft authored by a group at Argonne National Laboratory [GLLO85]. But, to be honest, we found that manuscript even harder to understand than Warren's report. The flaw was that it insisted in presenting the complete WAM as is, rather than as a gradually transformed and optimized design.

A gradual refinement style has in fact been used by David Maier and David S. Warren[2] in [MW88]. There, one can find a description of techniques of Prolog compilation akin to the WAM's.[3] However, we believe that this otherwise quite commendable effort still suffers from a few drawbacks as a definitive tutorial. First, it describes a close variant of the WAM rather than, strictly speaking, the WAM itself. That is, not all of the WAM's features are covered. Moreover, explanations are limited to illustrative examples and seldom make explicitly and exhaustively clear the specific context in which some optimizations apply. Second, the part devoted to compilation

[1]David H. D. Warren's confides privately that he "felt [that the WAM] was important, but [its] details unlikely to be of wide interest. Hence, [he used a] 'personal notes' style."

[2]A different person than the WAM's designer, for whose research the WAM has been of great inspiration. In turn, interestingly enough, David H. D. Warren has lately been working on a parallel architecture for Prolog whose abstract model shares its essence with some ideas independently conceived by David S. Warren.

[3]*Op. Cit.*, Chapter 11.

of Prolog comes very late in the book—in the penultimate chapter—relying, for implementation details, on overly detailed Pascal procedures and data structures incrementally refined over the previous chapters. We feel that this sidetracks reading and obfuscates to-the-point learning of the abstract machine. Finally, although it presents a series of gradually refined designs, their tutorial does not separate orthogonal pieces of Prolog in the process. All the versions presented are full Prolog machines. As a result, the reader interested in picking and choosing partial techniques to adapt somewhere else cannot discriminate among these easily. Now, in all fairness, Maier and Warren's book has the different ambition of being a first course in logic programming. Thus, it is actually a feat that its authors were able to cover so much material, both theoretical and practical, and go so far as to include also Prolog compiling techniques. More important, their book is the first available official publication to contain a (real) tutorial on the WAM techniques.

After the preliminary version of this book had been completed, another recent publication containing a tutorial on the WAM was brought to this author's attention. It is a book due to Patrice Boizumault [Boi88] whose Chapter 9 is devoted to explaining the WAM. There again, its author does not use a gradual presentation of partial Prolog machines. Besides, it is written in French—a somewhat restrictive trait as far as its readership is concerned. Still, Boizumault's book is very well conceived, and contains a detailed discussion describing an explicit implementation technique for the *freeze* meta-predicate.[4]

Even more recently, a formal verification of the correctness of a slight simplification of the WAM was carried out by David Russinoff [Rus89]. That work deserves justified praise as it methodically certifies correctness of most of the WAM with respect to Prolog's SLD resolution semantics. However, it is definitely not a tutorial, although Russinoff defines most of the notions he uses in order to keep his work self-contained. In spite of this effort, understanding the details is considerably impeded without working familiarity with the WAM as a prerequisite. At any rate, Russinoff's contribution is nevertheless a *première* as he is the first to establish rigorously something that had been taken for granted thus far. Needless to say, that report is not for the fainthearted.

[4] *Op. Cit.*, Chapter 10.

1.2 This tutorial

1.2.1 Disclaimer and motivation

The length of this monograph has been kept deliberately short. Indeed, this author feels that the typical expected reader of a tutorial on the WAM would wish to get to the heart of the matter quickly and obtain complete but short answers to questions. Also, for reasons pertaining to the specificity of the topic covered, it was purposefully decided not to structure it as a real textbook, with abundant exercises and lengthy comments. Our point is to make the WAM explicit as it was conceived by David H. D. Warren and to justify its workings to the reader with convincing, albeit informal, explanations. The few proposed exercises are meant more as an aid for understanding than as food for further thoughts.

The reader may find, at points, that some design decisions, clearly correct as they may be, appear arbitrarily chosen among potentially many other alternatives, some of which he or she might favor over what is described. Also, one may feel that this or that detail could be "simplified" in some local or global way. Regarding this, we wish to underscore two points: (1) we chose to follow Warren's original design and terminology, describing what he did as faithfully as possible; and, (2) we warn against the casual thinking up of alterations that, although that may appear to be "smarter" from a local standpoint, will generally bear subtle global consequences interfering with other decisions or optimizations made elsewhere in the design. This being said, we did depart in some marginal way from a few original WAM details. However, where our deviations from the original conception are proposed, an explicit mention will be made and a justification given.

Our motivation to be so conservative is simple: our goal is *not* to teach the world how to implement Prolog optimally, *nor* is it to provide a guide to the state of the art on the subject. Indeed, having contributed little to the craft of Prolog implementation, this author claims glaring incompetence for carrying out such a task. Rather, this work's intention is to explain in simpler terms, and justify with informal discussions, David H. D. Warren's abstract machine *specifically* and *exclusively*. Our source is what he describes in [War83, War88]. The expected achievement is merely the long overdue filling of a gap so far existing for whoever may be curious to acquire *basic* knowledge of Prolog implementation techniques, as well as to serve as a spring board for the expert eager to contribute further to this field for which the WAM is, in fact, just the tip of an iceberg. As such, it is hoped that this monograph would constitute an interesting and self-contained complement to basic textbooks for general courses on logic programming, as well as to those on compiler design for more conventional programming languages. As a stand-alone work, it could be a quick reference for the computer professional in need of direct access to WAM concepts.

1.2.2 Organization of presentation

Our style of teaching the WAM makes a special effort to consider carefully each feature of the WAM design in isolation by introducing separately and incrementally distinct aspects of Prolog. This allows us to explain as limpidly as possible specific principles proper to each. We then stitch and merge the different patches into larger pieces, introducing independent optimizations one at a time, converging eventually to the complete WAM design as described in [War83] or as overviewed in [War88]. Thus, in Chapter 2, we consider unification alone. Then, we look at flat resolution (that is, Prolog without backtracking) in Chapter 3. Following that, we turn to disjunctive definitions and backtracking in Chapter 4. At that point, we will have a complete, albeit *naïve*, design for pure Prolog. In Chapter 5, this first-cut design will be subjected to a series of transformations aiming at optimizing its performance, the end product of which is the full WAM. We have also prepared an index for quick reference to most critical concepts used in the WAM, something without which no (real) tutorial could possibly be complete.

It is expected that the reader already has a basic understanding of the operational semantics of Prolog—in particular, of unification and backtracking. Nevertheless, to make this work also profitable to readers lacking this background, we have provided a quick summary of the necessary Prolog notions in Appendix A. As for notation, we implicitly use the syntax of so-called Edinburgh Prolog (see, for instance, [CM84]), which we also recall in that appendix. Finally, Appendix B contains a recapitulation of all explicit definitions implementing the full WAM instruction set and its architecture so as to serve as a complete and concise summary.