

Quo Vadis Answer Set Programming?

Past, Present, and Future

V.W. Marek

Department of Computer Science
University of Kentucky
Lexington, KY 40506-0046, USA

Abstract. We discuss the development, current state and the future of Answer Set Programming, making predictions that are not necessarily accurate.

1 How did it happen, or prehistory

In early 1984, less than a year after I joined CSD, University of Kentucky, my former student, late Witold Lipski suggested that I look at nonmonotonic logic. It will be *next big thing*, he wrote and suggested reading papers by Raymond Reiter. I went a step further and went to the first Nonmonotonic Logic Workshop in Mohonk, NY. That workshop was, in fact, co-organized by Reiter. Fate had it - the organizers made me share the room with Vladimir Lifschitz who immediately straightened up my wrong ideas on nonmonotonic logic. I knew that change of semantics from all models to a subclass may result in a nonmonotonic consequence operation; this phenomenon was observed by professional logicians earlier. But what I did not know was that the idea was to formalize (fragments of) commonsense reasoning.

That same year another fortunate event of my scientific life occurred – Mirek Truszczyński joined our department. Two years later we understood what autoepistemic logic was really about and what the algorithms for manipulation of modal formulas in autoepistemic context were. In 1987 Michael Gelfond visited us in Lexington and talked about the use of autoepistemic logic to provide semantics for negation in logic programming [Gel87]; the goal that at that time appeared elusive. Then, in 1988 we got the seminal paper of Gelfond and Lifschitz [GL88]. I presented it at our Logic and AI seminar. Very quickly we understood that the technique was closely related to that of Reiter (via appropriate fixpoint construct). Putting it all together, Mirek and I realized that the complexity of basic problems related to stable semantics can be established. It all came together quickly.

We looked at the implementation of stable semantics almost immediately, although we did not know what this can do for us. Eric and Elisabeth Freeman were our students at the time and they implemented a very simple stable semantics solver (no bells and whistles). While I cannot find the written report of that work (but there was one), my recollection is that it could handle programs with 30 variables at most. Mirek and I then

moved to a more ambitious project called DeReS (Default Reasoning System). Today forgotten, it was an implementation of Reiter's logic and thus stable semantics as well. Two Ph.D. students, Paweł Cholewiński and Artur Mikitiuk worked with Mirek and me on that project. By 1993, the area was mature enough so Mirek and I could publish a monograph of the results so far [MT93].

It was Ilkka Niemelä who, like Mirek and I, came to stable semantics through autoepistemic logic. He made a breakthrough in the implementation of stable semantics with his *smodels*. What was very important in that research and implementation was the first grounder, *lparse*, built in collaboration with Patrik Simons. Another important system, *d1v* was started in Vienna and continued in Vienna and Calabria. This one was a solver for the disjunctive logic programming; a powerful extension of stable semantics.

In 1991, Anil Nerode, V.S. Subrahmanian and I started the conference series called *Logic Programming and Nonmonotonic Reasoning*, alternating every other year with Nonmonotonic Logic Workshop. This action created a community which exists today.

In 1998 Mirek and I, and independently Ilkka realized that stable model semantics can be used *efficiently and user-friendly* for solving NP-search problems. By pure coincidence that same year Krzysztof Apt wanted to have a workshop for 25 years of logic programming (and an accompanying volume) and we, in Lexington, had a venue: the beautiful Pleasant Hill, a place where a (now extinct) religious group of Shakers built its paradise on earth. Mirek and I presented the idea there. A catchy name was needed and was provided by Vladimir.

2 What do we have now

When we met at Stanford in 2001 for the first ASP workshop, the name was there and the issue was how better and faster ASP solvers could be built. Bart Selman was asked for an invited presentation and he talked about the progress with SAT solvers, a technology which in the past number of years made a tremendous leap forward based on several theoretical advances, better data structures and more careful implementations. The class of problems solved by SLP solvers and SAT solvers is exactly the same. But SAT solvers were significantly faster (not uniformly, but on average). Not only systems such as *sato* and *grasp* but truly lightning-speed systems like *chaff* were showing the way forward. Both *smodels* and *d1v* used improvements that were suggested by the SAT community, but not everything that the SAT community used was directly applicable. A new idea was needed and the next big thing in ASP was the appearance of ASSAT, the system built by Fangzhen Lin and Yuting Zhao. Its novel idea was based on a careful studies of reasons why some supported models are not stable. In the hindsight, one could see that it was all about the cycles in the call graph (and since Apt, Blair and Walker, and then later work by Stefania Costantini and Alessandro Provetti we knew that there was a connection). But the issue was that while the completion of the program can easily be reduced to a propositional theory without significant increase in the size of the resulting theory, adding the clauses "killing" the cycles increases the size of the corresponding theory exponentially. Fortunately, it turned out that this can be done piecemeal, maybe killing *some* cycles (this was called by Lin and Zhao *adding*

loop formulas) was enough. There are, unfortunately exponentially many loop formulas (as explained in [LR06]), but not all of them have to be there to find stable model.

I am sure that there are various ways ASSAT and other systems based on the idea of loop formulas can be viewed. For me ASSAT and *cmodels* are examples of the approach that has been successful in many other areas, namely so-called *Satisfiability modulo theories* (SMT) where a back-end SAT engine is used as a generator and enumerator for *candidate assignment*. In this approach (very successful in a variety of applications in electronic design automation [NOT04], but also in other areas, for instance Lintao Zhang and Ilya Mironov SAT attack on hash function collisions), a propositional theory generating some assignments is used to assist the programmer in systematic search for solution. The domain specific engine (in this case checker that tests if the assignment indeed a stable model) is used as a front-end. Thus, SAT is used as a back-end enumerator, and (possibly optimized) checker is used as the front-end, neatly tying ASP to SAT.

May be it is just a sign of times, but a new thing in this period was the appearance of ASP competition and a benchmarking environment *Asparagus*. It looks like it is no longer possible to talk about solving without actually solving it.

The pioneering work of Niemelä resulted in extending ASP to the context where constructs such as cardinality constraints and weight constraints can be used within the programs. This extended significantly the conciseness of knowledge representation with ASP and should result in better usability of the ASP in practice. We need new types of constraints that can be added to stable semantics to facilitate the tasks of programmers.

3 Where to from here

There is, as of now, very little commercial development of ASP. The system closest to the use as a non-academic, “for-profit” software is, clearly, `d1v` (of course I may not be aware of all that is available). There are several reasons for this situation. In my mind, the most important issue is the lack of easy explanation of stable semantics to a “programmer from the street”. While the Computer Engineering students can (and even often are) taught about SAT, I suspect that it would be quite difficult to explain to a *general audience* the semantics of logic programs. I tried this in the past, and failed (below Ph.D. students’ level, of course). There are many equivalent descriptions of stable semantics (viz. recent text by Lifschitz [Li08]), but we should somehow make it explainable on undergraduate, or beginning graduate level. It is even more difficult to explain disjunctive logic programming (or am I just inept?). This, in my mind, is the most important stumbling block for applicability of ASP in electronic design automation, the area driving progress in SAT. This is an important issue: once the *initial investment* in understanding stable semantics is made, the advantages of stable semantics are obvious. To give one example (due to Mirek) implementing Cannibals and Missionaries puzzle in ASP is easy; doing it directly with SAT is unpleasant. That is knowledge representation with ASP is much easier, and we should take advantage of this phenomenon.

The question of software engineering for ASP still needs answers and tools. There were meetings and papers, but we still are far away from production-level environments,

or even understanding what are those problems that software engineering is supposed to handle in this area.

Another problem is the working with databases. The `dlv` system is a nice exception; for a number of years now they paid attention to the issues of getting data from and to databases. It is not difficult, and in principle ASP systems can even simulate SQL, but, of course this should not be done, database systems do this better.

Yet another issue is the availability of data types. I do not mean only the basic issues such as availability of string manipulation (important with database applications) but it is possible to define type systems over user defined types (as done within extensional database). As long as reasonable limitations are imposed (for instance length of available lists is specified) this can be done. To the best of my knowledge such constructs were not studied in the ASP context. But may be I am a pessimist here. Adding XML constructs (as suggested by Thomas Eiter in his work on Semantic Web) may do the same thing.

In recent years there was a lot of work on various forms of equivalence of programs. This clearly has a software engineering flavor (like *does my subroutine always do the same thing as your subroutine*). In my mind, however strong equivalence is something else. Namely, it attempts to find a *correct* logic for stable logic semantics (and more generally, disjunctive logic programming). The connections of logics such as intuitionistic logic and programs were observed early. The most amazing thing of strong equivalence is the use of the maximal intermediate logic HT (Gödel-Smetanich logic)[LPV01]. It is surprising that this logic appears in the context of software engineering of ASP. Whether it really will take us somewhere beyond theoretical progress – remains to be seen.

As we progress with the theory of ASP we need a better bounds on both the number of answer sets and on the bounds on time needed to find first solution [LT06].

The work of Niemelä on constraints [NSS99] were followed by many (myself and Jeff Remmel included). But we still do not have a definitive account of the treatment of constraints in ASP. It is, in fact quite important, as (in my view) successful implementation of those constraints resulted in the increased interest in some of these constraints in SAT (of course 0-1 Integer Programming is another candidate for the source of this influence).

Generally, since the very beginning (and this is the legacy of Logic Programming), the importance of Knowledge Representation with ASP was of primary interest to all of us. This is different from the attitude of the sister community, SAT, where these issues were not so prominent (although, of course SAT planning is, primarily an issue in knowledge representation). Several extensive articles and even a book [GL02,Ba03] were devoted to this issue in ASP. This all needs to be seen in the major context mentioned at the beginning; the problems with explaining of ASP to the wider community of potential users, especially in electronic design automation. It is not enough to use ASP (say for model checking, [TT07]), the issue is to convince others to use it. For that reason, for instance `dlv` offers a front-end which uses the solver as a back-end engine.

Time for conclusions. As is clear from my presentation, I believe that a simpler, clearer, less technical descriptions of ASP must be found. Being by nature an optimist, I believe that they will be found. The elegance of knowledge representation with ASP will then open the possibility of a wider use of ASP. I also believe that one way or

another we are bound to get closer with SAT community. Signs of this phenomenon are already seen. So, I do not know how the celebrations of the 40th anniversary of stable semantics of logic programs will look like. But I am sure there will be a lot to celebrate.

References

- [Ba03] Baral, C., *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press, 2003.
- [Gel87] Gelfond, M. On stratified autoepistemic theories. *Proceedings of AAAI 1987*, pages 207–211, 1987.
- [GL88] Gelfond, M. and Lifschitz, V. The stable model semantics for logic programming. In *Proceedings of the International Joint Conference and Symposium on Logic Programming*. MIT Press, 1070–1080, 1988.
- [GL02] Gelfond, M. and Leone, N. Logic Programming and Knowledge Representation – A-Prolog perspective. *Artificial Intelligence* 138:3–38, 2002.
- [Li08] Lifschitz, V., Twelve Definitions of a Stable model. This volume.
- [LPV01] Lifschitz, V., Pearce, D., and Valverde, A. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2:526–541, 2001.
- [LR06] Lifschitz, V., and Razborov, A., Why are there so many loop formulas? *ACM Transactions on Computational Logic* 7:261-268, 2006.
- [LZ04] Lin, F. and Zhao, Y. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence Journal* 157:115–137, 2004.
- [LT06] Lonc, Z. and Truszczyński, M., Computing minimal models, stable models and answer sets. *Theory and Practice of Logic Programming* 6:395-449, 2006.
- [MT93] Marek, V.W., and Truszczyński, M.: *Nonmonotonic Logic; Context-Dependent Reasoning*. Springer, Berlin, 1993.
- [MT99] Marek, V. and Truszczyński, M., Stable Models and an Alternative Logic Programming Paradigm. *The Logic Programming Paradigm*, pp. 375–398. Series Artificial Intelligence, Springer-Verlag, 1999.
- [Nie99] Niemelä, I. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25:241–273, 1999
- [NSS99] Niemelä, I., Simons, P., and Soininen, T., Stable Model Semantics of Weight Constraint Rules. *Proceedings of LPNMR 1999*, pages 317-331, 1999.
- [NOT04] Nieuwenhuis, R., Oliveras, A., and Tinelli, C., Abstract DPLL and Abstract DPLL Modulo Theories. *Proceedings of LPAR 2004*, pages 36-50, 2004.
- [TT07] Tang, C. and Ternovska, E. Model Checking Abstract State Machines with Answer Set Programming. *Fundamenta Informaticae* 77:105-141, 2007.