Making your hands dirty inspires your brain! Or how to switch ASP into Production Mode

Torsten Schaub*

Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89,
D-14482 Potsdam, torsten@cs.uni-potsdam.de

Rising from strong theoretical foundations in Logic Programming and Nonmonotonic Reasoning, Answer Set Programming (ASP) came to life as a declarative problem solving paradigm [1,2,3] in the late nineties. The further development of ASP was greatly inspired by the early availability of efficient and robust ASP solvers, like *smodels* [4] and *dlv* [5]. The community started modeling with ASP and a first milestone was the conception of TheoryBase [6] providing a systematic and scalable source of benchmarks stemming from combinatorial problems. Although the scalability of such benchmarks is of great value for empirically evaluating systems, the need for application-oriented benchmarks was early perceived. The demand for systematic benchmarking led to the Dagstuhl initiative and with it the creation of the web-based benchmark archive *asparagus* [7]. This repository has in the meantime grown significantly, mainly due to the two past ASP competitions [8,9], and contains nowadays a whole variety of different types of benchmarks, although it is still far from being comprehensive.

Meanwhile, the prospect of ASP has been demonstrated in numerous application scenarios.[1] A highlight among them is arguably the usage of ASP for the high-level control of the space shuttle [10]. What makes this application so special is the fact that it was solving an application problem in a real-world environment. Although we still need many more elaborated proofs of concept, showing how ASP addresses different application scenarios, solving such real(-world) problems is yet another issue. Let me approach this by answering some preliminary questions.

What is a real problem? Such a problem could be an unsolved combinatorial or mathematical problem. Also, it could stem from an application that is traditionally solved with different methods. In either case, the problem is not academic anymore, but rather about producing an effective solution. To accomplish this, we have to switch to a *production mode*,[2] that is, the process of organizing the production of a solution to truly challenging problems. This mode of operation goes (currently) quite beyond conceptual modeling and benchmarking that most of us are used to so far.

Why should we solve real problems? Apart from the fact that the prospect of doing so partly nourishes our right of existence, real problems are a tremendously fruitful source of new research questions. For instance, concepts like cardinality and weight constraints [11], magic set transformations [12], constraint additions to ASP [13,14], or projection [15], had never been pushed so hard without a real problem driving their development. In other words, making our hands dirty inspires our brain!

---

* Affiliated with Simon Fraser University, Canada, and Griffith University, Australia.

[1] See http://www.kr.tuwien.ac.at/research/projects/WASP and/or http://www.cs.uni-potsdam.de/~torsten/asp for an overview.

[2] This term goes back to Marxist theory and has been adopted in informatics in various ways.

Where do we find real problems? One source are open problems known to academia. For instance, deciding the fifth Schur number.[3] In fact, the benchmarking repository of the Automated Theorem Proving community used to contain unsolved challenge problems that were a driving force for the development of theorem provers. And after all, they made it into the New York Times by proving Robbins' conjecture.[4] Actually, interesting challenge problems are often simply down the hall, sitting on your colleagues' desks. Getting interested in their problems, makes them discover ASP and exhibits us to non-artificial problems. The least expected result is a proof of concept or a benchmark suite, the greatest is to make a difference with ASP!

How do we solve real problems? To begin with, one still starts with a proof of concept addressing a toy version of the actual problem. If this scales and the solution is satisfactory, one just hit the jackpot. Unfortunately, this never happens (to me) and brings us back to the aforementioned production mode of ASP. The dilemma is that we must address real problems in order to further develop ASP as a tool for addressing real applications. This vicious cycle makes current production processes far from ideal and dominated by pragmatics, and often not even addressable by means of ASP only.

The first bottleneck in the ASP production mode is the encoding. This has become a true art and often the initial, rather declarative problem specification bears little resemblance with the final stream-lined encoding reducing combinatorics. This also applies to automatically generated encodings. This is not to say that the final encoding is not indicative but it needs quite some experience to be produced. Moreover, the optimization of encodings is also tightly connected to the target ASP system, and in particular, its grounding component. It makes quite a difference whether the grounder relies on domain predicates or not, and whether it provides special-purpose methods, like constraint handling techniques or unification for avoiding grounding large domains. Clearly, this track leaves the idea of declarative problem solving behind and the burden of optimizing encodings has to be partly taken off the user and handled (semi-)automatically in the long run.

The second bottleneck is the configuration of the actual ASP solver. Modern ASP solvers relying on Boolean constraint technology offer a manifold arsenal of parameters for controlling the search for answer sets. For instance, *clasp* has roughly forty options [16], half of which control the search strategy. Choosing the right parameters often makes the difference between being able to solve a problem or not. But again this takes us away from the idea of declarative problem solving and automatic methods must be conceived for partially relieving this second burden.[5]

As a matter of fact, the two aforementioned bottlenecks are not regarded as problematic in the SAT community. Rather encodings are often presented to a solver at its convenience and industrial problems are solved with particular parameter settings (eg. aggressive restart strategies). This marks a true difference in the philosophy of both communities.

---

[3] Thanks to Mirosław Truszczyński for pointing this out.

[4] `http://www.nytimes.com/library/cyber/week/1210math.html`

[5] For instance, a first such prototype is *claspfolio* using machine learning techniques for mapping problem features to solver parameters.

Finally, it is surprising that the lack of software engineering tools is yet not even an issue in ASP's production mode. The reason is simply that the production mode in ASP is up to now accomplished by experts in ASP and no end users. This lack will become a true bottleneck once ASP would principally be ready for real applications.

So, let's make our hands dirty and get inspired!

References

1. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3-4) (1999) 241–273
2. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In Apt, K., Marek, W., Truszczyński, M., Warren, D., eds.: The Logic Programming Paradigm: a 25-Year Perspective. Springer (1999) 375–398
3. Lifschitz, V.: Answer set planning. In de Schreye, D., ed.: Proceedings of ICLP'99, The MIT Press (1999) 23–37
4. Niemelä, I., Simons, P.: Evaluating an algorithm for default reasoning. In: Working Notes of the IJCAI'95 Workshop on Applications and Implementations of Nonmonotonic Reasoning Systems. (1995) 66–72
5. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: A deductive system for nonmonotonic reasoning. In Dix, J., Furbach, U., Nerode, A., eds.: Proceedings of LPNMR'97. Springer (1997) 363–374
6. Cholewiński, P., Marek, V., Mikitiuk, A., Truszczyński, M.: Experimenting with nonmonotonic reasoning. In Sterling, L., ed.: Proceedings of ICLP'95, The MIT Press (1995) 267–281
7. Borchert, P., Anger, C., Schaub, T., Truszczyński, M.: Towards systematic benchmarking in answer set programming: The Dagstuhl initiative. In Lifschitz, V., Niemelä, I., eds.: Proceedings of LPNMR'04. Springer (2004) 3–7
8. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In Baral, C., Brewka, G., Schlipf, J., eds.: Proceedings of LPNMR'07. Springer (2007) 3–17
9. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczyński, M.: The second answer set programming competition. [17] To appear.
10. Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., Barry, M.: An A-prolog decision support system for the space shuttle. In Ramakrishnan, I., ed.: Proceedings of PADL'01. Springer (2001) 169–183
11. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
12. Faber, W., Greco, G., Leone, N.: Magic sets and their application to data integration. Journal of Computer and System Sciences **73**(4) (2007) 584–609
13. Mellarkod, V., Gelfond, M.: Integrating answer set reasoning with constraint solving techniques. In Garrigue, J., Hermenegildo, M., eds.: Proceedings of FLOPS'08. Springer (2008) 15–31
14. Gebser, M., Ostrowski, M., Schaub, T.: Constraint answer set solving. In Hill, P., Warren, D., eds.: Proceedings of ICLP'09. Springer (2009) 235–249
15. Gebser, M., Kaufmann, B., Schaub, T.: Solution enumeration for projected Boolean search problems. In van Hoeve, W., Hooker, J., eds.: Proceedings of CPAIOR'09. Springer (2009) 71–86
16. Gebser, M., Kaufmann, B., Schaub, T.: The conflict-driven answer set solver clasp: Progress report. [17] To appear.
17. Erdem, E., Lin, F., Schaub, T., eds.: Proceedings of LPNMR'09. Springer (2009)