Künstliche Intelligenz manuscript No.
(will be inserted by the editor)

Answer Set Programming Unleashed!

Torsten Schaub · Stefan Woltran

Received: date / Accepted: date

Abstract Answer Set Programming faces an increasing popularity for problem solving in various domains. While its modeling language allows us to express many complex problems in an easy way, its solving technology enables their effective resolution. In what follows, we detail some of the key factors of its success.

Answer Set Programming (ASP; [9]) is seeing a rapid proliferation in academia and industry due to its easy and flexible way to model and solve knowledge-intense combinatorial (optimization) problems. To this end, ASP offers a high-level modeling language paired with high-performance solving technology. As a result, ASP systems provide out-off-the-box, general-purpose search engines that allow for enumerating (optimal) solutions. They are represented as answer sets, each being a set of atoms representing a solution. The declarative approach of ASP allows a user to concentrate on a problem's specification rather than the computational means to solve it. This makes ASP a prime candidate for rapid prototyping and an attractive tool for teaching key AI techniques since complex problems can be expressed in a succinct and elaboration tolerant way. This is eased by the tuning of ASP's modeling language to knowledge representation and reasoning (KRR). The resulting impact is nicely reflected by a growing range of successful applications of ASP [19,20].

Torsten Schaub has been supported by the German Science Foundation (DFG): projects SCHA 550/9 and 11. Stefan Woltran has been supported by the Austrian Science Fund (FWF): project Y698.

T. Schaub
University of Potsdam E-mail: torsten@cs.uni-potsdam.de

S. Woltran
TU Wien E-mail: woltran@dbai.tuwien.ac.at

Model, ground, and solve

ASP is often designated as a model-ground-and-solve paradigm due to its work flow, in which a high-level first-order specification is turned by a grounder into a propositional format that is then used by a solver to compute (optimal) solutions of the original problem.

The modeling methodology of ASP follows a guess-and-check approach: solution candidates are delineated and tested for feasibility [24]. This may yield none, one, or multiple answer sets representing solutions. The modeling language of ASP builds upon rules featuring first-order variables, function symbols, and various forms of aggregates [3]. In addition, quantitative and qualitative preferences can be specified for optimization [43,8]. This results in an expressive high-level language that distinguishes ASP from other solving paradigms. Meanwhile, the core of this language has been standardized [11]. The first-order nature of ASP's input language lets us separate the actual problem encoding from its problem instances. Encodings are expressed via first-order rules and instances as facts. This separation enables meta-programming and leaves us with an interesting margin. Take a block-world planning example. The vanilla way is to encode blocks-world planning and fix the specific blocks and locations in the instance. A more suitable approach is to encode planning as such and consider the blocks-world domain along with its specifics as an instance. In the most extreme case, one may even encode ASP (or modifications thereof) per se and consider the given (reified) logic program as a problem instance, which amounts to full-fledged meta-programming.

ASP grounding is concerned with the systematic replacement of first-order variables by (admissible) terms [29]. To this end, it relies upon techniques from semi-naive database evaluation [1]. In fact, grounding a single rule amounts to view materialization, just as with SQL, or alternatively, to solving a constraint satisfaction problem. This already gives a glimpse on the computational complexity of grounding, which is in general even Turing-complete (due to the usage of function symbols). Thanks to the aforementioned standardization efforts, grounders have become off-the-shelf systems with well-defined input and output formats. This constitutes another distinguishing feature of ASP.

The core technology used for ASP solving builds upon techniques originally developed for Satisfiability Testing (SAT; [7]). At first, ASP systems relied upon DPLL[1] techniques [43,31], whereas modern systems are based upon Conflict-Driven Clause Learning (CDCL; [34, 44]) [23,2]. While the performance of modern ASP solvers is at eye-height with today's SAT solvers, ASP systems are much more versatile in order to support complex KRR tasks [29]. On the one hand, this is induced by the need to support ASP's rich modeling language, including aggregates and optimization. On the other hand, other forms of reasoning need to be supported beyond satisfiability checking, namely, enumeration, projection, intersection, union, etc. of solutions. The versatility of its solvers constitutes yet another distinguishing feature of ASP.

Foundations

Unlike other modeling languages, ASP comes with solid logical foundations. As first observed by Pearce [39], ASP rests upon the (monotonic) logic of here-and-there [27], a constructive logic featuring a genuine implication connective. This connective captures the semantic essence of ASP rules. Interestingly, negation is defined in terms of this implication, which contrasts the historic misconception that ASP is built upon the concept of negation-as-failure. Answer sets are defined as distinguished models in the logic of here-and-there (characterized by a minimality criterion) as detailed in [10].

This simple logical characterization is in a certain contrast to the origin of ASP. In fact, early definitions of its semantics relied on fixpoint characterizations [6, 26] reflecting intuitions from neighboring areas like Nonmonotonic Logics, Logic Programming, and Datalog. Research on ASP at that time mainly focused on its relationship to other formalisms. For instance, ASP can, on the one hand, be regarded as a restricted fragment of default logic [41] (cf. [26]), or, on the other hand, as an alternative approach to Prolog for treating negation-as-failure. Furthermore, answer sets can also be characterized by means of second-order logic very similar to the characterization of minimal models in Circumscription [35] (cf. [40]). Many other characterizations can be found in the literature and their different intuitions rely on the respective context or purpose. In particular, in recent years, computational aspects gained importance and steered research towards further characterizations, most notably in terms of completion, loop formulas and variants thereof, see [13,33].

Cosmos

Prolog [14] has been the most influential approach to employing logic as a programming language. The original semantics of ASP was conceived for characterizing Prolog programs. At that time, logic programs were expected to have at most one answer set. However, the different treatment of loops over negation resulted in a deviation from this principle. This led to the aforementioned paradigm of ASP in which different solutions are represented by a multitude of answer sets. In fact, Prolog should be seen as a programming language whose execution amounts to proof search via SLD resolution [30]. This execution model is fully transparent to the user. This is in stark contrast to ASP, whose problem resolution is completely opaque and bestows ASP a much more declarative flavor than Prolog.

Evidently, ASP is not the only approach for solving combinatorial (optimization) problems. Related paradigms are SAT, Constraint Processing (CP; [17]), and (Mixed) Integer/Linear Programming (MILP; [42]). Closest to ASP is SAT and its various extensions for MAXSAT, PB, and QBF solving [7]. All of them offer highly performant solving technology but fall short in supporting KRR in its full spectrum. Foremost, the unavailability of mod-

[1] Davis–Putnam–Logemann–Loveland [16,15]

eling languages forces users of SAT-based technology to embed their problem encodings in compilers. This makes them non-transparent, and hard to modify and to maintain. In John McCarthy's words, such approaches lack elaboration-tolerance [36]. A further consequence of the propositional input format of SAT-based technology is that problem encodings and problem instances are inseparable.

Another major difference lies in the distinct semantic foundations. While ASP relies on a non-monotonic semantics in the spirit of closed-world reasoning, all aforementioned SAT-based approaches build upon monotonic logics and thus on open-world reasoning. These foundations allow ASP to model many concepts frequently encountered in KRR problems in a fairly easy way. This includes (un)reachability (and transitive closures), inertia (for solving frame problems), defaults, etc. Such concepts are quite tricky to handle in a monotonic setting. This representational edge can be pinned down by two formal results. First, SAT can be compiled into ASP in a modular way but not vice versa [37]. Second, compiling ASP to SAT may lead to an exponential blow-up, unless the language is extended [32]. From a technological point of view, ASP can however be regarded as an instance of Satisfiability Modulo Theories (SMT; [7]) whose theory reasoning accounts for the additional inferences of ASP.

CP and MILP rely also upon monotonic foundations. Unlike ASP and SAT, however, their focus lies on constraints over integer or even real-valued variables.[2] To accommodate such non-Boolean constraints, ASP and SAT offer hybrid reasoning techniques that integrate the best of both worlds [5,28].

A prominent representative of closed-world reasoning are database systems. As mentioned, database queries can be represented as ASP rules. The latter extend the database machinery by unbounded recursion. More precisely, ASP constitutes a proper extension of Datalog [12].

Challenges

Despite the success of ASP, nontrivial challenges remain.

True declarativity remains the holy grail. ASP is declarative in strictly separating logic from control. However, two equivalent encodings may still exhibit significant performance differences. This calls for sophisticated methods for source code optimization.

ASP needs more software engineering methodologies and tools. For instance, the strict separation of logic and control makes traditional debugging techniques inapplicable. Also, only few development environments exist.[3]

Finally, ASP has to be fit for laymen. ASP attracts more and more users outside of computer science due to its declarative- and effectiveness. To consolidate its applicability by laymen, a simplified user language could enter ASP standardization efforts.

Conclusion

ASP has come a long way. Having its roots in Nonmonotonic Logics, Logic Programming and Databases, as well as Satisfiability Testing, the ASP community can be proud of having taught Tweety how to fly: It has produced robust declarative systems for KRR by building upon solid formal foundations.

Resources

– Textbooks and tutorials: [4,18,22,25]
– ASP language standard: [11]

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: F. Calimeri, G. Ianni, M. Truszczyński (eds.) Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15), *Lecture Notes in Artificial Intelligence*, vol. 9345, pp. 40–54. Springer-Verlag (2015)
3. Alviano, M., Faber, W.: Aggregates in answer set programming. Künstliche Intelligenz (2018). To appear.
4. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
5. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability modulo theories. In: Biere et al. [7], chap. 26, pp. 825–885
6. Bidoit, N., Froidevaux, C.: Minimalism subsumes default logic and circumscription in stratified logic programming. In: Proceedings of the Second Annual Symposium on Logic in Computer Science (LICS'87), pp. 89–97. IEEE Computer Society Press (1987)

[2] Note that problems over real-valued variables only are not combinatorial and solvable in polynomial time.

[3] Exceptions include [38,21].

7. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, *Frontiers in Artificial Intelligence and Applications*, vol. 185. IOS Press (2009)
8. Brewka, G., Delgrande, J., Romero, J., Schaub, T.: asprin: Customizing answer set preferences without a headache. In: B. Bonet, S. Koenig (eds.) Proceedings of the Twenty-Ninth National Conference on Artificial Intelligence (AAAI'15), pp. 1467–1474. AAAI Press (2015). URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9535`
9. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
10. Cabalar, P., Pearce, D., Valverde, A.: Answer set programming from a logical point of view. Künstliche Intelligenz (2018). To appear.
11. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2: Input language format. Available at `https://www.mat.unical.it/aspcomp2013/ASPStandardization` (2012)
12. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer-Verlag (1990)
13. Clark, K.: Negation as failure. In: H. Gallaire, J. Minker (eds.) Logic and Data Bases, pp. 293–322. Plenum Press (1978)
14. Clocksin, W., Mellish, C.: Programming in Prolog. Springer-Verlag (1981)
15. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5**, 394–397 (1962)
16. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7**, 201–215 (1960)
17. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
18. Eiter, T., Ianni, G., Krennwallner, T.: Answer Set Programming: A Primer. In: S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, R. Schmidt (eds.) Fifth International Reasoning Web Summer School (RW'09), *Lecture Notes in Computer Science*, vol. 5689, pp. 40–110. Springer-Verlag (2009). URL `http://www.kr.tuwien.ac.at/staff/tkren/pub/2009/rw2009-asp.pdf`
19. Erdem, E., Gelfond, M., Leone, N.: Applications of ASP. AI Magazine **37**(3), 53–68 (2016)
20. Falkner, A., Friedrich, G., Schekotihin, K., Taupe, R., Teppan, E.: Industrial applications of answer set programming. Künstliche Intelligenz (2018). To appear.
21. Febbraro, O., Reale, K., Ricca, F.: ASPIDE: Integrated development environment for answer set programming. In: J. Delgrande, W. Faber (eds.) Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11), *Lecture Notes in Artificial Intelligence*, vol. 6645, pp. 317–330. Springer-Verlag (2011)
22. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers (2012)
23. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artificial Intelligence **187-188**, 52–89 (2012)
24. Gebser, M., Schaub, T.: Modeling and language extensions. AI Magazine **37**(3), 33–44 (2016)
25. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press (2014)
26. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: R. Kowalski, K. Bowen (eds.) Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88), pp. 1070–1080. MIT Press (1988)
27. Heyting, A.: Die formalen Regeln der intuitionistischen Logik. In: Sitzungsberichte der Preussischen Akademie der Wissenschaften, p. 42–56. Deutsche Akademie der Wissenschaften zu Berlin (1930). Reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.
28. Kaminski, R., Schaub, T., Wanko, P.: A tutorial on hybrid answer set solving with clingo. In: G. Ianni, D. Lembo, L. Bertossi, W. Faber, B. Glimm, G. Gottlob, S. Staab (eds.) Proceedings of the Thirteenth International Summer School of the Reasoning Web, *Lecture Notes in Computer Science*, vol. 10370, pp. 167–203. Springer-Verlag (2017)
29. Kaufmann, B., Leone, N., Perri, S., Schaub, T.: Grounding and solving in answer set programming. AI Magazine **37**(3), 25–32 (2016)
30. Kowalski, R.: Predicate logic as programming language. In: Proceedings IFIP Congress, pp. 569–574. North-Holland Publishing Company (1974)
31. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3), 499–562 (2006)
32. Lifschitz, V., Razborov, A.: Why are there so many loop formulas? ACM Transactions on Computational Logic **7**(2), 261–268 (2006)
33. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 112–118. AAAI/MIT Press (2002)
34. Marques-Silva, J., Sakallah, K.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**(5), 506–521 (1999)
35. McCarthy, J.: Applications of circumscription to formalizing common-sense knowledge. Artificial Intelligence **28**, 89–116 (1986)
36. McCarthy, J.: Elaboration tolerance (1998). URL `http://www-formal.stanford.edu/jmc/elaboration.html`
37. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3-4), 241–273 (1999)
38. Oetsch, J., Pührer, J., Tompits, H.: Catching the ouroboros: On debugging non-ground answer-set programs. Theory and Practice of Logic Programming **10**(4-6), 513–529 (2010)
39. Pearce, D.: Equilibrium logic. Annals of Mathematics and Artificial Intelligence **47**(1-2), 3–41 (2006)
40. Pearce, D., Tompits, H., Woltran, S.: Encodings for equilibrium logic and logic programs with nested expressions. In: P. Brazdil, A. Jorge (eds.) Proceedings of the

Tenth Portuguese Conference on Artificial Intelligence (EPIA'01), *Lecture Notes in Computer Science*, vol. 2258, pp. 306–320. Springer-Verlag (2001)

41. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13**(1-2), 81–132 (1980)
42. Schrijver, A.: Theory of linear and integer programming. Discrete mathematics and optimization. John Wiley & sons (1999)
43. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2), 181–234 (2002)
44. Zhang, L., Madigan, C., Moskewicz, M., Malik, S.: Efficient conflict driven learning in a Boolean satisfiability solver. In: R. Ernst (ed.) Proceedings of the International Conference on Computer-Aided Design (ICCAD'01), pp. 279–285. IEEE Computer Society Press (2001)