Modern Constraint Answer Set Solving

Max Ostrowski[1]

[1]University of Potsdam, Germany

1 Introduction

Answer Set Programming (ASP;[4]) is a declarative problem solving approach, combining a rich yet simple modeling language with high-performance solving capabilities using techniques from Satisfiability Checking (SAT;[20, 7]). This has already resulted in various applications, among them decision support systems for NASA shuttle controllers [22, 2], product configuration [27], scheduling [17], timetabling [3], shift design [1] and various reasoning tools in systems biology [5, 11, 15, 23, 10, 24]. However, certain aspects of such applications are more naturally modeled by additionally using non-Boolean propositions, accounting for resources, fine timings, or functions over finite domains. Moreover, a dedicated treatment of large domains avoids the grounding bottleneck that is, the need to discretize large domains, inherent to all propositional solving approaches.

In SAT, this led to the subarea of Satisfiability Modulo Theories (SMT;[21]), extending SAT solvers by theory-specific solvers for arithmetic, arrays, finite sets, bit vectors, equality with uninterpreted functions etc. It allows SMT problems to incorporate predicates from specialized theories into propositional formulas. Solving an SMT problem consists of finding a (hybrid) assignment to all Boolean and theory-specific variables satisfying a given formula along with its theory-specific constituents. Apart from a close solver integration, the key to efficient SMT solving lies in elaborated conflict-driven learning techniques that are capable of combining conflict information from different solver types (cf. [21]).

To be able to handle resources and quantities within ASP, I concentrate on one theory and extend ASP with constraints for integer arithmetics. This paradigm is called Constraint Answer Set Programming (CASP). My goal is to extend the modeling language of ASP with constraints while preserving its declarative nature. This allows for fast prototyping and elaboration tolerant problem descriptions. Furthermore, I want to preserve the raw processing speed of the underlying inference engine.

Groundbreaking work on enhancing ASP with Constraint Processing (CP;[9, 25]) techniques was conducted in [6, 18, 19]. Based on firm semantical underpinnings, this approach provides a family of ASP languages parameterized by different constraint classes. While [6] develops a high-level algorithm viewing both ASP and CP solvers as black boxes, [19] embeds a black-boxed CP solver into a traditional DPLL-style backtracking algorithm, similar to the one underlying the ASP solver *smodels* [26]. Although [6, 18, 19] resulted in two consecutive extensions of *smodels* with CP capabilities, they do not match the performance of state of the art SMT solvers, simply because they cannot take advantage of

elaborated conflict-driven learning techniques. I address this problem and propose several alternative ways to combine ASP and CP solving. My thesis will therefore capture the following topics.

2 Constraint Answer Set Programming via Conflict Driven Nogood Learning

To define CASP, I pursue a semantic approach that is based on a propositional language rather than a multi-sorted, first-order language, as used in [6, 18, 19]. This allows to use conflict-driven nogood learning (CDNL;[14]) technology for solving propositional formulas. These learning algorithms are the state of the art solution to any satisfiability problem and have been well researched since the mid-90s. I use and extend these sophisticated algorithms for solving CASP problems. My approach follows the so-called lazy approach of advanced SMT solvers by abstracting from the constraints in a specialized theory [21]. The idea is as follows. During solving, the ASP solver passes its (partial) knowledge to a CP solver, which checks the implied constraints against its theory via constraint propagation. As a result, it either signals that no solution exists or, if possible, extends the knowledge base of the ASP solver. To facilitate learning within the ASP solver, however, each inference must be justified, providing a "reason" for the underlying algorithms. Yet, to the best of my knowledge, this is not supported by off-the-shelf CP solvers.[1]

I show the correctness of my approach by proving the relation between the definition of CASP and its characterization using nogoods. As a consequence, I develop an algorithmic framework for conflict-driven ASP solving that integrates CP solving capabilities while overcoming the aforementioned difficulty. An implementation named *clingcon* is presented, outperforming previous approaches. It is able to handle optimization functions over constraint variables and global constraints. In a second step, the algorithmic framework is extended by filtering techniques based on irreducible inconsistent sets (IIS;[29, 16]). This technique strengthens the provided conflicts and improves the learning capabilities of the whole approach.

3 Encoding Constraint Satisfaction Problems

For solving Constraint Satisfaction Problems (CSPs), the preferred method is not so clear and new approaches developed during the last years. Having a standard, non-learning CP solver has the benefit of supporting special (global) constraint propagators for various kinds of constraints. An implicit variable/domain representation supports huge or even infinite domains. Encoding finite linear CSPs as propositional formulas and solving them by using modern solvers for SAT has proven to be a highly effective approach by the award-winning *sugar*[2] system.

[1] Advanced SMT solvers, like [21], address this through handcrafted theory solvers.
[2] http://bach.istc.kobe-u.ac.jp/sugar

The CP solver *sugar* reads a CSP instance and transforms it into a propositional formula in Conjunctive Normal Form (CNF). The translation relies on the order encoding [8, 28], and the resulting CNF formula can be solved by an off-the-shelf SAT solver. I elaborate upon an alternative approach based on ASP and present the resulting *aspartame*[3] framework. It constitutes an ASP-based CP solver similar to *sugar*. The major difference between *sugar* and *aspartame* rests upon the implementation of the translation of CSPs into Boolean constraint problems. While *sugar* implements a translation into CNF in Java, *aspartame* starts with a translation into a set of facts. These facts are combined with a general-purpose ASP encoding for CP solving (also based on the order encoding). Extending the used techniques, I define CASP using nogoods and provide an ASP library for solving it.

4 Lazy Nogood and Variable Creation

The first approach used to handle CASP consists of a learning ASP solver in combination with a non-learning CP solver. Without learning facilities, these CP solvers rest upon an implicit variable representation. It permits huge domains and avoids the grounding bottleneck, but also restricts information exchange which impedes the CDNL algorithm. The presented translation approach, encoding CASP using ASP, explicitly represents integer variables and therefore benefits from the full power of CDNL. The granularity induced by this representation provides accurate conflict and propagation information. On the other hand, it limits scalability due to the size of the translation. I therefore present an approach, combining the use of CDNL with an explicit representation, overcoming the named weaknesses of the two approaches. I use dedicated propagators to implicitly represent the encoding of the constraints and create the necessary nogoods and variables whenever needed. This means that we neither need to make the constraints nor the variables explicit a priori, but create them on demand. In combination with a generic, declarative theory language and sophisticated preprocessing techniques I provide a full fledged implementation of a modern CASP solver, named *clingcon* 3. I evaluate my system and compare it with state of the art CP and CASP solvers, thereby providing a tool for translating CP benchmarks in the *minizinc* format into the internal ASP format. This enables the CASP community to take advantage of a whole new class of benchmarks.

5 Multi-Shot Constraint Answer Set Programming

Multi-shot ASP solving [12, 13] is about solving continuously changing logic programs in an operative way. This can be controlled via reactive procedures that loop on solving while reacting, for instance, to outside changes or previous solving results. These reactions may entail the addition or retraction of rules that the operative approach can accommodate by leaving the unaffected program parts

[3] https://potassco.org/labs/2016/09/20/aspartame.html

intact within the solver. This avoids re-grounding and benefits from heuristic scores and constraints learned over time. Evolving constraint logic programs can be extremely useful in dynamic applications to add new resources, set observed variables, and add or relieve restrictions on capacities. To extend multi-shot solving to CASP, *clingcon* 3 is able to add and delete constraints in order to capture evolving CSPs. New resources can be added using additional constraint variables and domains. While restricting variables by adding constraints and rules to the constraint logic program is easy, increasing their capacity is not. The key to this is lazy variable creation, to avoid making huge domains explicit. For this purpose, I start with a virtually maximum domain that is restrained by retractable constraints. The domain is then increased by relaxing these constraints. This avoids introducing a large amount of atoms. I exemplify this approach using the well known n-queens and the yale shooting problem.

References

1. M. Abseher, M. Gebser, N. Musliu, T. Schaub, and S. Woltran. Shift design with answer set programming. *Fundamenta Informaticae*, 147(1):1–25, 2016.
2. M. Balduccini, M. Gelfond, and M. Nogueira. Answer set based design of knowledge systems. *Annals of Mathematics and Artificial Intelligence*, 47(1-2):183–219, 2006.
3. M. Banbara, K. Inoue, B. Kaufmann, T. Schaub, T. Soh, N. Tamura, and P. Wanko. teaspoon: Solving the curriculum-based course timetabling problems with answer set programming. In E. Burke, L. Di Gaspero, B. McCollum, A. Schaerf, and E. Özcan, editors, *Proceedings of the Eleventh International Conference of the Practice and Theory of Automated Timetabling (PATAT'16)*, pages 13–32, 2016.
4. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
5. C. Baral, K. Chancellor, N. Tran, N. Tran, A. Joy, and M. Berens. A knowledge based approach for representing and reasoning about signaling networks. In *Proceedings of the Twelfth International Conference on Intelligent Systems for Molecular Biology/Third European Conference on Computational Biology (ISMB'04/ECCB'04)*, pages 15–22. Oxford University Press, 2004.
6. S. Baselice, P. Bonatti, and M. Gelfond. Towards an integration of answer set and constraint solving. In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.
7. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
8. J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In B. Hayes-Roth and R. Korf, editors, *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 1092–1097. AAAI Press, 1994.
9. R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
10. M. Durzinsky, W. Marwan, M. Ostrowski, T. Schaub, and A. Wagler. Automatic network reconstruction using ASP. *Theory and Practice of Logic Programming*, 11(4-5):749–766, 2011.

11. S. Dworschak, T. Grote, A. König, T. Schaub, and P. Veber. Tools for representing and reasoning about biological models in action language $\mathcal{C}$. In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 94–102, 2008.
12. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Clingo* = ASP + control: Extended report. Technical report, Universität Potsdam, 2014.
13. M. Gebser, R. Kaminski, P. Obermeier, and T. Schaub. Ricochet robots reloaded: A case-study in multi-shot ASP solving. In T. Eiter, H. Strass, M. Truszczyński, and S. Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation: Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Artificial Intelligence*, pages 17–32. Springer-Verlag, 2015.
14. M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89, 2012.
15. M. Gebser, T. Schaub, S. Thiele, B. Usadel, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 130–144. Springer-Verlag, 2008.
16. J. Gleeson and J. Ryan. Identifying minimally infeasible subsystems of inequalities. In *ORSA Journal On Computing*, volume 2, pages 61–63. Operations Research Society of America, 1990.
17. G. Grasso, S. Iiritano, N. Leone, V. Lio, F. Ricca, and F. Scalise. An ASP-based system for team-building in the Gioia-Tauro seaport. In M. Carro and R. Peña, editors, *Proceedings of the Twelfth International Symposium on Practical Aspects of Declarative Languages (PADL'10)*, volume 5937 of *Lecture Notes in Computer Science*, pages 40–42. Springer-Verlag, 2010.
18. V. Mellarkod and M. Gelfond. Integrating answer set reasoning with constraint solving techniques. In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.
19. V. Mellarkod, M. Gelfond, and Y. Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):251–287, 2008.
20. D. Mitchell. A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science*, 85:112–133, 2005.
21. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
22. M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An A-prolog decision support system for the space shuttle. In I. Ramakrishnan, editor, *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer-Verlag, 2001.
23. M. Ostrowski, G. Flouris, T. Schaub, and G. Antoniou. Evolution of ontologies using ASP. In J. Gallagher and M. Gelfond, editors, *Technical Communications of the Twenty-seventh International Conference on Logic Programming (ICLP'11)*,

volume 11, pages 16–27. Leibniz International Proceedings in Informatics (LIPIcs), 2011.
24. M. Ostrowski, L. Paulevé, T. Schaub, A. Siegel, and C. Guziolowski. Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming. *Biosystems*, 149:139–153, 2016.
25. F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier Science, 2006.
26. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.
27. T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In G. Gupta, editor, *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL'99)*, volume 1551 of *Lecture Notes in Computer Science*, pages 305–319. Springer-Verlag, 1999.
28. N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.
29. J. van Loon. Irreducibly inconsistent systems of linear inequalities. *European Journal of Operational Research*, 8(3):283–288, 1981.