Marina De Vos (Editor)

LPNMR 2017 Doctoral Consortium

Hanasaari Conference Center, Espoo, Finland

4-6 July 2017

Preface

The 14th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR) is a forum for exchanging ideas on declarative logic programming, non-monotonic reasoning, and knowledge representation. The aim of the conference is to facilitate interactions between researchers and practitioners interested in the design, implementation and application of logic-based programming languages and database systems, and those who work in the area of knowledge representation and non-monotonic reasoning. LPNMR and its programmatic expression, Answer Set Programming, have roots in the famous special issue of AIJ in 1980, devoted to Nonmonotonic Reasoning.

The LPNMR Doctoral Consortium (DC) runs in parallel to LPNMR 2017 giving participants a chance to experience the main conference as well as the DC. The event takes place in the Hanasaari Conference Center, Espoo, Finland, July 4-6, 2017. The DC consists of two presentation sessions, a poster session and a lunch with mentors from across the field. It gives students the opportuni to present and discuss their research and to obtain feedback from peers as well as world-renowned experts a supportive environment.

This volume contains the eight papers presented at LPNMR DC 2017.

As doctoral consortium chair, I would like to thanks the LPNMR programme co-chairs, Marcello Balduccini and Tomi Janhunen for their support in making this event possible and allowing it run alonside LPNMR 2017. I would also like to thanks the programme committee members for their carefull reviewing and constructive feedback. Many thanks for the DC mentors for your invaluable advise to the next generation of LPNMR researchers.

June 12, 2017 — Marina De Vos
Bath — Doctoral Consortium Chair

Table of Contents

Program Committee

Pedro Cabalar	University of Corunna
Marina De Vos	University of Bath
Esra Erdem	Sabanci University
Yuliya Lierler	University of Nebraska at Omaha
Alessandra Mileo	Dublin City University, INSIGHT Centre for Data Analytics
Julian Padget	University of Bath
Alessandra Russo	Imperial College London
Hans Tompits	Vienna University of Technology
Stefan Woltran	TU Wien

Sampling and Search Space with Answer Set Programming

Flavio Everardo

University of Potsdam, Germany

Abstract. Answer Set Programming (ASP) is a declarative problem solving paradigm oriented on solving complex (high) combinatorial problems in an efficient way. Depending on the problem, an ASP Solver can compute a large set of answers and being difficult to track a desired answer among the search space. For this kind of problems, it can be infeasible to know all the search space and computing all answers may complicate more the idea of asking for "the best" answer or at least one desired model for the user. Saying this, ASP solvers can be extended to new reasoning modes to pursue sampling or search space navigation. Addressed to this problematic, my research focus on sampling and search space using ASP. The goal is to apply them on diverse real-life problems which such as Music Production and Urban Planning, both powered with ASP. Finally, I present related work as well as challenges and ideas to develop.

1 Introduction

Answer Set Programming (ASP;[2]) is a declarative problem solving paradigm oriented on solving complex combinatorial problems in an efficient way. ASP is based on a simple yet expressive rule language that allows to easily model (describe) problems in a compact and intuitive form. ASP has become very popular in areas, which involve problems of combinatorial search using a considerable amount of information to process like Automated Planning, Robotics, Linguistics, Biology, Narratology, Content Generation for Video Games and even Music [15] [16] [17]. Saying this, exist real life problems (in areas like Computer Music or Urban Planning, to name a few) that there is not a single or even a small amount of answers. Instead there are several (over thousands or millions) possible and valid answers for a single problem. A single or a set of results given by a State-of-the-art ASP solver cannot fulfill your requirements for that specific time and asking for a new model over and over again may not be an option. This is the reason why ASP solvers can be extended to new reasoning modules to pursue sampling or search space navigation and let the user to see different parts of all the spectrum of answers. The goal of my research is to explore several ways to extend ASP with sampling and search space to contribute in the ASP domain and in further developments inside areas such as Urban Planning and Computer Music Production work using ASP among others. This extended abstract is divided in two main sections, being the first one the work related to sampling, search space and lazy grounding, followed by the application of this research into the musical and urban planification.

2 Sampling, Search Space and Lazy Grounding

Computing through a very high dimensional space is an unsolved problem of scientific computation which can be used in several applications [10], but as the volume grows the problem can become computationally intractable. This phenomenon is known as the curse of dimensionality [10] and previous efforts have worked on sampling techniques. These techniques includes probabilistic reasoning including XOR (parity) constraints generation and hash functions with polynomial calls to SAT solvers, which has also worked for model counting. On the other hand, [11] has demonstrated parity reasoning modules to extend the capabilities of SAT solvers through unit propagation or equivalence reasoning. This approach does not mention sampling as a final use, just as circuit verification or logical cryptanalysis, but it can be adapted work as a sampling generator. Inside ASP a novel approach for computing diverse (or similar) solutions to logic programs using preferences [12] is developed taking the advantages of multi-shot ASP solving [18]. This framework can be used for Design Space Exploration [13] to find representative Pareto optimal solutions. The combination of different approaches can derive into new methods for sampling with uniform distribution and new reasoning modes can be added as part of the built-in options from a State-of-the-art ASP solver as *clingo* [1]. This effort may lead to seach the space for this kind of problems and change the direction of the search by adding new (auto-generated) constraints or sampling rules on the fly, with an incremental solving perspective like *iclingo* [19]. Additionally, for an user interaction perspective, another technique for study is the use of relevant literals, selected by the user to guide the search for the desired solution(s) [20]. The goal is to be able to compute in advance several rules and/or constraints, so with or without user interaction the reasoning mode can derive a better search and a more well-distributed sampling.

3 Computer Music and Urban Planning

The success and establishment of ASP has opened new opportunities to explore real life problems and scenarios in music. So far ASP is able to compose diverse types of musical styles [3] [4] including lyrics that matches the composition [9]. Other works proposes to build and compose chords progressions and cadences [6] [7], create score variations [5] or fill the spaces to complete a composition [8]. Finally, a worked that I submitted to the Sound and Music Computing (SMC) Conference to be organized from July 5 – 8, 2017 in Espoo, Finland is about using ASP to propose a multitrack balanced mix for studio-music (post) production engineering area. A knowledge base is compiled with rules and constraints extracted from the literature about what professional music producers and audio engineers suggest creating a good mix leaving the solutions generation to ASP.

This kind of problems can lead to a huge amount of results, such as over 300,000 possible configurations to compose one single music measure or even over 2000 different results to mix just 6 instruments in stereo. As mentioned before sampling and search space will play a key role in the musical domain because of its subjective nature. Also it is difficult to categorize an answer as good or bad if it respects satisfiability but navigate and explore a few set of answers extracted in a smart way can lead the user to a better experience and getting a suitable answer for her/his needs. Inside Urban Planning, something similar happens. The unorganized growth of cities brings significant environmental issues that affect the quality of life of its inhabitant. This kind of problem can contain several variables that may derive into countless results. A previous effort using ASP [14] has shown that it is possible to extend this problems into a declarative language with the goal to make recommendations or modifications for urban areas with the purpose of finding alternatives to fulfill a proper land use. Currently there are evaluations ongoing as an alternative to narrow the number of results without sampling. Examples are maximizing the proper use of lands and use of natural resources mixed with minimization of traveling costs, edification needed and wastes but still the number of results are difficult to handle.

References

1. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and P. Wanko. Theory solving made easy with clingo 5. In OASIcs-OpenAccess Series in Informatics. Vol. 52. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
2. M. Gebser, R. Kaminski, B. Kaufmann and T. Schaub. *Answer Set Solving in Practice*, Morgan and Claypool Publishers, 2012.
3. Boenn, Georg and Brain, Martin and De Vos, Marina and Ffitch, John. Automatic music composition using answer set programming, Theory and practice of logic programming 11.2-3 (2011): 397-427.
4. Everardo, Flavio and Aguilera, Antonio. Armin: Automatic Trance Music Composition using Answer Set Programming, Fundamenta Informaticae, volume 113 number 1, 2011. IOS Press.
5. Flavio Everardo. A Logical Approach for Melodic Variations., LA-NMR. 2011, pages 141-150
6. Opolka, Sarah and Obermeier, Philipp and Schaub, Torsten. Automatic Genre-Dependent Composition using Answer Set Programming, Proceedings of the 21st International Symposium on Electronic Art. 2015.
7. Eppe, Manfred and Confalonieri, Roberto and Maclean, Ewen and Kaliakatsos, Maximos and Cambouropoulos, Emilios and Schorlemmer, Marco and Codescu, Mihai and Kühnberger, Kai-Uwe Computational invention of cadences and chord progressions by conceptual chord-blending, 2015.
8. Rodrigo Martín-Prieto Herramienta para armonización musical mediante Answer Set Programming, University of Corunna, Spain. 2016.
9. Toivanen, Jukka Mikael et.al. Methods and Models in Linguistic and Musical Computational Creativity, Helsingin yliopisto. 2016.
10. Sabharwal, Ashish, and Bart Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization., 2013.

11. Laitinen, Tero. Extending SAT solver with parity reasoning., 2014.
12. Romero, Javier, Torsten Schaub, and Philipp Wanko. Computing Diverse Optimal Stable Models., OASIcs-OpenAccess Series in Informatics. Vol. 52. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
13. Philipp Wanko. Scalable Design Space Exploration via Answer Set Programming., OASIcs-OpenAccess Series in Informatics. Vol. 52. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
14. Munoz, Jennifer, and Flavio Everardo. Urban Land Use Planning using Answer Set Programming., LANMR 2016.
15. Gelfond, Michael, and Yulia Kahl. Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach., Cambridge University Press, 2014.
16. Smith, T., Padget, J., and Vidler, A. Design Space Descriptions for Logical Generation of Content., IOS Press, 2016.
17. Thompson, M., Padget, J., and Battle, S. Governing narrative events with institutional norms., 2015.
18. Gebser, M., Obermeier, P., and Schaub, T. Interactive Answer Set Programming-Preliminary Report. arXiv preprint arXiv:1511.01261, 2015.
19. M. Gebser and R. Kaminski and B. Kaufmann and M. Ostrowski and T. Schaub and S. Thiele. Engineering an Incremental ASP Solver In International Conference on Logic Programming (pp. 190-205). Springer Berlin Heidelberg, 2008.
20. Athakravi, D., Corapi, D., Russo, A., De Vos, M., Padget, J., and Satoh, K. Handling change in normative specifications. In International Workshop on Declarative Agent Languages and Technologies (pp. 1-19). Springer Berlin Heidelberg, 2012.

Structure-Driven Answer-Set Solving*

Markus Hecher

TU Wien, Austria; University of Potsdam, Germany
hecher@dbai.tuwien.ac.at

Abstract Parameterized algorithms are a way to solve hard problems efficiently, given that a specific parameter of the input is small. A well-studied parameter is treewidth, which roughly measures to which extent the structure of a graph resembles a tree. In our research, we want to exploit this parameter in the context of Answer Set Programming. In the literature, algorithms have been proposed that are linear in the program size assuming bounded treewidth. However, this approach works only in case of small treewidth. In consequence, we aim at new methods "between" traditional techniques and algorithms exploiting structural parameters.

Keywords: Tree Decompositions, Dynamic Programming, Fixed-Parameter Tractability, Answer-Set Programming, Parameterized Complexity

1 Introduction

Parameterized algorithms [14,8] have attracted considerable interest in recent years and allow to tackle hard problems by directly exploiting a small parameter of the input problem. One particular goal in this field is to find guarantees that the runtime is exponential exclusively in the parameter. A structure-based parameter that has been researched extensively is treewidth [15,5]. Generally speaking, treewidth measures the closeness of a graph to a tree, based on the observation that problems on trees are often easier than on arbitrary graphs. A parameterized algorithm exploiting small treewidth typically takes a *tree decomposition* (TD), which is an arrangement of a graph into a tree, and evaluates the problem in parts by *dynamic programming* (DP) on the TD.

Answer Set Programming (ASP) [6] is a logic-based declarative modelling language and problem solving framework where solutions, so called answer sets, of a given logic program directly represent the solutions of the modelled problem. ASP has been successfully applied in several application domains and industrial needs, which, in particular, accelerates the search for alternative solving methods exploiting major aspects of relevant instances, as for example the structure of these instances. This generally brings up the question whether structure-based parameters as for instance treewidth aids in evaluating logic programs. Jakl et al. [12] give a DP algorithm that is linear in the input size of the program, double exponential in the treewidth of a certain graph representing the program

* Research was supported by the Austrian Science Fund (FWF), Grants Y698, P25607.

structure, and restricted to disjunctive rules. In our work [10], we presented an extension on how to evaluate extended logic programs based on the full ASP-Core-2 syntax. However, this paradigm seems to be only of use in case of small, bounded treewidth. In consequence, the idea is to study alternative evaluation techniques somewhere "between" traditional algorithms for ASP and algorithms exploiting structure-based parameters of logic programs. This research topic enables a broad range of specialization, ranging from the study of parameterized algorithms to CDCL-based algorithms exploiting structure of logic programs.

2 Preliminaries

Tree Decompositions (TDs) & Parameterized Algorithms. TDs form a tool to capture essential parts of the structure of a given graph instance. These decompositions are trees consisting of nodes, which contain (parts of) the given graph instances. The so-called "width" of such a TD is essential for the definition of the parameter "treewidth", which captures in a sense, how hard the given graph instance is when solving a certain problem. The smaller the treewidth of a given graph, the more "tree-like" the graph and thus typically "easier" to solve problems on the graph. TDs provide a way on how to tackle hard problems by evaluating a certain input problem instance (represented as a graph) in parts, thereby being sensitive to the treewidth of the instance. Of particular interest are so-called *fixed-parameter tractable* (FPT) algorithms with respect to a certain parameter k, which are capable of evaluating instances such that the runtime depends polynomially on the instance size and on $f(k)$ for some computable function f. If the parameter k is reasonably small, such an algorithm seems preferable compared to an approach, which requires exponential runtime in the worst-case. In consequence, such FPT algorithms with respect to treewidth perform well for certain instances and are typically based on *dynamic programming* (DP) on TDs, which computes parts of the problem obtained by iterating a TD of the problem instance, and combines them accordingly. There are also open-source systems like *D-FLAT* [3], which provide a general DP framework. Several techniques incorporated in our *DynASP* solver [10,9] for ASP, stem from D-FLAT.

Answer Set Programming (ASP). ASP is typically evaluated by two involved components, namely (i) the grounder and (ii) the solver. The grounder is responsible for producing a ground program during the *grounding*, a process which eliminates variables in an originally *non-ground* program containing variables. One can think of such non-ground programs as general sets of rule schemes, whereas ground programs are obtained by instantiating these general rule schemes into actual ASP rules. The main interest of this research proposal concerns part (ii), i.e., how to efficiently evaluate a set of ASP rules forming a *logic program*. However, note that in practice the performance of ASP techniques not only require efficient solving techniques, but also rely on smart grounding tools. Hence, especially the bridge between grounder and solver is of interest and might lead to further investigations concerning the structure of programs. As already mentioned, there also exist FPT algorithms for solving logic programs [10,12] by means of TDs.

3 Proposed Research

In this section, we first discuss the aim covered in this proposal and then give a detailed description of the results obtained so far and planned research tasks.

3.1 Research Aim

Gutin emphasized the necessity of establishing [1] parameterized algorithmics. As it turned out (not unexpectedly), implementing theoretical ideas in a straightforward way does not immediately yield practically successful systems. Several obstacles for properly handling real-world instances need to be taken into account, as we will discuss also in Section 3.2. We therefore identify the following aims:

– Incorporate concepts exploiting structure into existing solvers (and probably grounders) for ASP in order to make these tools more aware of the structure present in both data and rules, and to *improve solving performance*.
– Develop methods for building a novel, *competitive* ASP system consisting of (i) a solver that improves the DynASP system, and (ii) further methods exploiting and capturing structure of logic programs.
– Apply our findings in a broader, general context (for instance default logic).

3.2 Lessons Learned: Unexpected Results and Obstacles

The insights gained so far indicate that a general way to turn structure-driven answer set solving into a practical success is indeed challenging. First, we have observed that for ASP, structure can be fruitfully exploited. In other words, the treewidth of a ground program can be kept small when the instance has small treewidth as well (*structure in data*), but this depends on the actual (problem) encoding. Second, concerning the design of actual DP algorithms, we collected the following insights of the literature, which will also be the basis for my research.

– The shape of the TD on which DP is performed is crucial [1].
– A quite surprising result is that alternative space-efficient storage techniques via *Binary Decision Diagrams* (BDDs) can be extremely beneficial [7]. In fact, DP over TDs with widths up to 50 can be handled with such an approach.
– If it is enough to compute one solution, the classical, naive DP approach of computing all tables in their entirety before being able to print a solution can be substantially improved by a lazy-evaluation technique [4].

At the same time, it seems to be hard to compete with standard ASP solvers on the consistency problem (i.e., whether there is an answer set). In fact, we put much effort in the development of competitive systems, which excel mainly at counting answer sets, as we propose in [9]. We will thus aim (see Section 3.4) at ASP extensions that allow for reasoning over the sets of answer sets (similar to a result for monadic second-order logic [2]) and also develop new DP algorithms.

[1] G. Gutin: Should We Care about Huge Imbalance in Parameterized Algorithmics? *The Parameterized Compl. Newsletter* 11(2), `http://fpt.wdfiles.com/local--files/fpt-news:the-parameterized-complexity-newsletter/2015Dec.pdf`.

3.3 Obtained Results

The results obtained cover how structural properties can be exploited with focus on TDs and DP. First, we extended our DynASP solver [13] to the full ASP syntax as specified in the ASP-Core-2 standard. This includes interoperation with state-of-the-art grounders, and handling of all language constructs produced by such grounders such as weight constraints, optimization statements, choice rules and disjunctive rules. The implementation [10] works as follows: Ground ASP rules are represented as a graph and a TD is prepared, which is traversed in a bottom-up manner to evaluate the input program. We implemented several versions of such algorithms, based on the input program's graph representation, e.g., primal graphs, incidence graphs or a variant of incidence graphs where there is, in addition, a clique between weighted atoms (i.e., atoms in weight constraints or choice heads). Second, we improved the data structures in DynASP by using pointers to avoid duplicate data, as done in the D-FLATˆ2 [3] system. Our technical proposal [9] also includes a thorough experimental evaluation.

Given logic programs of small treewidth, our new ASP solver proved to be very competitive in the setting of model counting (#SAT), a central problem in areas like machine learning, statistics, probabilistic reasoning and combinatorics. When counting answer sets (#ASP), our approach has a big advantage: It does not need to materialize the full answer sets in order to count them. This provided huge speed-ups against classical ASP systems like Clasp. However, our implementation was also able to beat SAT model counters like sharpSAT or Cachet, even though it was not specifically optimized for classical model counting. Benchmarks show that QBF model counting performance (comp. depQBF) is competitive as well.

3.4 Future Goals

Exploiting Treewidth in Existing Solvers. Our tasks concerning this goal cover extending CDCL-based ASP solvers like Clasp by exploiting structure of the input instances using suitable *heuristic parameterization.* Further, we aim at *improving branch-and-bound* algorithms used to solve an extended logic program capable of modelling cost optimization, by incorporating the instance structure.

A way to improve existing ASP technology is to push ASP solvers into the direction of DP on TDs by *modifying solver heuristics.* Heuristic modifications in ASP solvers have recently received increasing attention to solve problems in domains where dedicated solving strategies would have required rewriting problem encodings entirely. A main reason is due to the observation that solving problem instances efficiently by means of ASP solvers regularly relies on either a dedicated encoding, which integrates a certain heuristic or exploits certain structural properties of the problem, or controlling the solver heuristic explicitly. Our ultimate goal here is to develop methods for combining the two worlds of (i) state-of-the-art ASP solvers like Clasp and (ii) local approaches based on DP on TDs. We expect that integrating aspects of DP (either in terms of heuristics or by extending ASP solvers) can speed up CDCL-based ASP solvers on structured instances. A corresponding, very limited proof of concept is available.[2]

[2] `https://github.com/hmarkus/dynclasp`

ASP has been extended to also allow cost optimization, where solutions to such an extended logic program can be seen as "cost-minimal" answer sets. There exist algorithms to compute these solutions, which are based on branch-and-bound, and also other approaches. Branch-and-bound basically is a technique with the goal of searching for optimal answer sets in a systematic way by cutting off parts of the search space. However, there is no guarantee that the algorithm does not hit parts of an answer set, which might not be cost-optimal, several times during the search. One possible idea to overcome this limitation is to exploit the structure of these logic programs by *caching parts of answer sets* together with the corresponding cost found during the search in such a way, that the atoms of these parts reflect the content of some nodes of the TDs. In other words, whenever a new answer set is found, we store the answer set in parts such that each TD node remembers a table of parts of answer sets. After some potential solutions are proposed by the branch-and-bound algorithm, we obtain a tree of tables of answer set parts with the knowledge how these parts can be combined (by TD properties). In the end, the TD could enable us to combine these stored parts such that we might still obtain a valid answer set of lowest cost not found by the algorithm so far. Ultimately, the goal is to not derive answer sets, which contain *non-optimal* parts already contained in an answer set proposed before.

Building a Novel System. The goal is to develop a novel ASP system which makes direct use of the structure provided by the data and the program. Driven by recent success stories (see, e.g. [7,4]), the strategy is to *directly improve* DynASP's performance, and to *enhance* the system by certain features on the other hand.

The current implementation of DynASP provides (compared to Clasp) good results for solving ASP counting and enumeration problems in case of problem instances of sufficiently low treewidth, and is not yet competitive when solving the consistency problem. The reason is that DynASP does not apply preprocessing and constraint handling techniques and does not have a counterpart for conflict learning yet. Some of these disadvantages can be handled by integrating one or several *lightweight CDCL-solvers* for SAT or ASP into DynASP, which then solve the sub-problems induced by the TD nodes (during bottom-up solving), as implemented in our system D-FLAT [4]. Since the computation in each TD node works on an autonomic basis and only pushes new findings to the successor node(s), this leaves room for *improving the interplay* between these nodes. We believe that with the knowledge on how the (local) sub-programs evolve during the solving process (bottom-up traversal), there is a smart way to combine these lightweight solvers and improve their interplay by taking certain shortcuts. Especially *learning failing solution paths* could greatly improve the resulting solving performance. Learning might be promising in combination with lazy-evaluation [4], which proved valuable for optimization problems and aims at deriving answer sets without completely evaluating all the TD nodes.

Since DynASP turned out to be competitive for counting answer sets [10] (#ASP), our plan is to further improve by significantly reducing time spent on invalidating $\subset$-optimality of answer set candidates (see [9] for more details). One can easily identify instances with an exponential number (w.r.t. the treewidth) of ⊂-smaller model candidates and although this still suffices for the algorithm to be FPT, it is a performance bottleneck in practice. Recent approaches suggest *Binary Decision Diagrams* [7] with the benefits of compact representation.

Given the already explored correspondence between model counting and advanced reasoning tasks (e.g., Bayesian inference [16]), our future plan for DynASP includes a *query language for Bayesian reasoning* on top of it. This new language should be able to reduce queries to #ASP problems including, but not limited to "How plausible is it to assume that a given atom belongs to an answer set?", "If an atom is believed, what is the probability of another atom belonging to an answer set?" or "Is it safe to assume that one atom is more reasonable than an other atom?". Considerations also include solving Problog [11] programs.

References

1. M. Abseher, F. Dusberger, N. Musliu, and S. Woltran. Improving the Efficiency of Dynamic Programming on Tree Decompositions via Machine Learning. In *IJCAI*, pages 275–282. AAAI, 2015.
2. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *JAlg*, 12(2):308–340, 1991.
3. B. Bliem, G. Charwat, M. Hecher, and S. Woltran. D-FLATˆ2: Subset minimization in dynamic programming on tree decompositions made easy. *FI*, 147(1):27–61, 2016.
4. B. Bliem, B. Kaufmann, T. Schaub, and S. Woltran. ASP for Anytime Dynamic Programming on Tree Decompositions. In *IJCAI*, pages 979–986. AAAI, 2016.
5. H. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *TCJ*, 51(3):255–269, 2008.
6. G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
7. G. Charwat and S. Woltran. Dynamic programming-based QBF solving. In *QBF*, pages 27–40, 2016.
8. M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
9. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Answer Set Solving using Tree Decompositions and Dynamic Programming - The DynASP2 System -. Technical Report DBAI-TR-2016-101, TU Wien, 2016.
10. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Answer set solving with bounded treewidth revisited. In *LPNMR*, 2017. To appear.
11. D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *TPLP*, 15(3):358–401, 2015.
12. M. Jakl, R. Pichler, and S. Woltran. Answer-set programming with bounded treewidth. In *IJCAI*. AAAI, 2009.
13. M. Morak, N. Musliu, R. Pichler, S. Rümmele, and S. Woltran. A New Tree-Decomposition Based Algorithm for Answer Set Programming. In *ICTAI*, pages 916–918, 2011.
14. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. OUP, 2006.
15. N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *JAlg*, 7(3):309–322, 1986.
16. T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian Inference by Weighted Model Counting. In *AAAI*, pages 475–482. AAAI, 2005.

Answer Set Programs with External Source Access: Integrated Evaluation and New Applications

Tobias Kaminski

Institute of Information Systems, TU Wien, Vienna, Austria
kaminski@kr.tuwien.ac.at

Abstract. *Answer Set Programming* (*ASP*) [13] is a widely-used paradigm for declarative programming, where a problem is encoded by nonmonotonic rules and its solutions are extracted from the *stable models* of the corresponding *answer set program* computed by a solver. This paper summarizes my doctoral research on evaluation and applications of HEX-programs, which extend ASP with access to external sources. The focus is on integrating evaluation of external sources and solving by exploiting partial assignments, and introducing lazy-grounding.

1 Motivation

Due to current trends in distributed systems and information integration, there is an increasing need for accessing external information sources from within knowledge representation formalisms such as ASP. For instance, it might be necessary to integrate information derived from a (possibly remote) *Description Logic* (*DL*) ontology into the computation of an answer set. If the derivation in the ontology is relative to information in the ASP part, a bidirectional exchange between a DL reasoner and an ASP solver is required. This kind of interaction is not provided by ordinary ASP, and pre-computing all possible derivations from the ontology and adding them to the answer set program is often not feasible. Motivated by this, the HEX formalism [3] has been developed, where external sources can be referenced in a program, and are evaluated during solving. The approach is related to *SMT*, but the focus is more on techniques for evaluating general external sources represented by arbitrary computations, i.e. it enables an API-like approach s.t. a user can define plugins without expert knowledge on solver construction.

By employing HEX, a user can, e.g., define a library function for concatenating strings, accessed via an external predicate $\&concat$. It could be used as illustrated by the following rule, where a first name and a last name are provided, and $\&concat$ returns the full name: $\mathit{fullname}(\mathit{Full}) \leftarrow \&concat[F, L](\mathit{Full}), \mathit{firstname}(F), \mathit{lastname}(L)$.

HEX is very expressive since it enables a bidirectional exchange of information between a HEX program and an external source and thus, encompasses the formalization of nonmonotonic and recursive aggregates. Consequently, HEX is suited for a wide range of applications, but also requires sophisticated evaluation algorithms to deal with the complexity that goes along with the high expressiveness. For this reason, my thesis work aims at the design and implementation of novel solving techniques for improving the efficiency of the formalism in general, as well as for specific classes of programs. Another focus of my work is on new applications that leverage the provided techniques, and in turn push the advancement of the formalism.

2 Research Goals

Challenges regarding efficient evaluation of HEX programs comprise the lack of a tight integration of the solving process with the evaluation of external sources as well as with the grounding procedure. Accordingly, the main goals of my doctoral research are:

1. to design advanced reasoning techniques that improve the evaluation of HEX programs by tightly integrating processes which have so far been treated as mostly independent sub-problems.
2. to develop an innovative application of the HEX formalism that utilizes external atoms for integrating logic programming and probabilistic reasoning, and which can be employed for concrete use cases that require reasoning over complex relational as well as uncertain information.
3. to implement new techniques that are developed for the evaluation algorithm within the DLVHEX solver for HEX programs [5], and to empirically investigate their performance by evaluating them on benchmark problems.

3 Background

Here, I start by summarizing the work most related to HEX and its applications, and I briefly introduce the theoretical background which my thesis work is based on.

Related Work. As there are many scenarios where it is more natural, and often more efficient, to outsource some information or computation in ASP, several approaches exist for this purpose, realizing different degrees of integration. DLV-EX programs [2] represent an early approach, which enables bidirectional communication with an external source, and allows the introduction of new terms by *value invention*. The CLINGO system also provides a mechanism for importing the extension of user-defined predicates via function calls during grounding [12]. In both cases, the interaction is more restricted than in HEX such that, e.g., nonmonotonic aggregates cannot be expressed.

CLINGO 5 [11] provides generic interfaces for combining theory solving with ASP, but its semantics differs from HEX and the approach is targeted at system developers. Besides, there are extensions of ASP towards the integration of specific sources; e.g., the CLINGCON system [17] implements *constraint ASP* relying on a tailored integration of a constraint solver. The setting of HEX differs as its goal is to enable a broad range of users to implement custom external sources and to harness efficient solving techniques.

HEX has been applied to a wide range of use cases. Among them are a framework for executing scheduled actions in external environments (*Act*HEX [10]); a system for merging belief sets based on nested HEX programs (*MELD system*, [18]); and an artificial agent able of playing the computer game *Angry Birds* (*Angry*HEX, [4]).

HEX Programs. HEX programs [3] extend answer set programs by allowing the use of *external atoms* of the form $\&g[p_1, ..., p_k](c_1, ..., c_l)$ in rule bodies, where $\&g$ is an *external predicate* name, $p_1, ..., p_k$ are input predicate names or constants, and $c_1, ..., c_l$ are output constants. The semantics of an external atom $\&g[p_1, ..., p_k](c_1, ..., c_l)$ is given by a *Boolean* $1 + k + l$-ary oracle function $f_{\&g}$ s.t. an external atom evaluates to *true* for a given assignment $\mathbf{A}$ over ordinary atoms if the oracle function returns *true*, i.e.

$f_{\&g}(\mathbf{A}, p_1, ..., p_k, c_1, ..., c_l) = \mathbf{t}$, and to *false* otherwise. The notion of satisfaction is extended to HEX rules and programs in the obvious way. Answer sets of a HEX program Π are those assignments $\mathbf{A}$ to ordinary atoms which are minimal models of the program consisting of all rules in Π of which the body is satisfied under $\mathbf{A}$ (the so-called *FLP-reduct* [9], an alternative to the well-known *GL-reduct*).

The basic procedure for computing the answer sets of a HEX program Π consists in replacing each (ground) external atom $\&g[p_1, ..., p_k](c_1, ..., c_l)$ by an ordinary atom of the form $e_{\&g[p_1,...,p_k]}(c_1, ..., c_l)$ and adding a guess $e_{\&g[p_1,...,p_k]}(c_1, ..., c_l) \vee ne_{\&g[p_1,...,p_k]}(c_1, ..., c_l) \leftarrow$ for its evaluation [6]. The result of this translation is an ordinary answer set program; and ordinary ASP solvers such as CLASP can be used for computing *model candidates*. However, guesses for external atoms must be checked afterwards for compatibility with the external semantics. By integrating *Conflict-Driven Nogood Learning* (*CDNL*) search into the HEX algorithm, the input-output relations can be learned from these checks in form of *nogoods* to avoid wrong guesses in the future search. Moreover, even if a model candidate complies with the answers of the corresponding oracle calls, it still needs to be checked for minimality wrt. the FLP-reduct.

4 Research Progress

In this section, I present the progress of my research since I started my doctoral studies.

Exploiting Partial Assignments for Efficient Evaluation of HEX Programs. Our work published in [8] concerns the semantics of external evaluations. Previously, oracle functions were only defined for complete inputs to external atoms, such that they could only be evaluated after the whole input was decided. As a result, many wrong guesses could only be detected late during search and nogoods were large because they usually entailed the complete input assignment. At the same time, this could not be improved when using two-valued assignments since external sources might be nonmonotonic, and they are *black boxes* such that theory specific techniques like in SMT cannot be applied.

We have overcome the mentioned challenges by extending the two-valued semantics to three-valued assignments that use the classical values *true* and *false*, and the new value *unassigned*. Based on partial assignments, we have introduced new evaluation techniques to increase the performance of the HEX algorithm. First, we have extended two-valued oracle functions to three-valued ones, which allows evaluation at any point during search under partial input. Second, nogoods now can also be learned under partial assignments, which are often significantly smaller. Moreover, given some input-output nogood, we obtain a set of minimal nogoods by applying a minimization procedure similar to the ones in [17]. The latter effectively speeds up the search since smaller nogoods potentially prune larger parts of the search space.

Naturally, there is a tradeoff between the runtime invested in additional oracle calls and the information gained from them. Thus, different evaluation heuristics are used for deciding whether an external source is evaluated at a point during search. The benefit of the new solving techniques has been verified by experiments using DLVHEX.

Lazy Grounding for HEX Programs. Our most recent work, which has been submitted for publication, considers the integration of a lazy grounding solver into the DLVHEX system, which exhibits promising results for classes of programs where the grounding bottleneck of ASP is an issue. This issue is even more challenging to tackle within the framework of HEX due to the need for grounding external atoms, which are largely black boxes from the viewpoint of the solver and may introduce new values. Lazy grounding avoids an exponential blowup of the grounding by interleaving grounding and solving, whereby rules are grounded on-the-fly depending on the satisfaction of their bodies. Nonmonotonic dependencies and value invention (i.e., import of new constants) from external sources make the integration nontrivial. As a result, we needed to introduce a novel external source interface to incrementally extend a HEX program grounding, where new output terms may appear *during solving*; and we have developed a novel evaluation algorithm for HEX programs that exploits a lazy grounding ASP solver.

Hybrid Classification of Visual Objects. As a starting point for investigating the potential and possible realization of combining ASP with probabilistic reasoning, I decided to consider a concrete application that requires logical as well as probabilistic reasoning. One of the basic tasks in *statistical relational learning* [14] is *collective classification*, which is simultaneously finding correct labels for a number of interrelated objects, for instance, predicting the classes of objects in a complex visual scene. Even if advanced algorithms for object recognition have been developed, they may fail unavoidably and yield ambiguous results due to few training data, noisy inputs, or inherent ambiguity of visual appearance. It is then still possible to draw on further information from the context in which an object occurs to disambiguate its label.

We have approached the collective classification problem in ASP by defining *hybrid classifiers* (*HC*) that combine local classifiers, which predict the probability of each local label given the features of a single object, with context constraints represented by weighted ASP constraints using object relations [7]. To obtain probabilistic semantics, we have embedded it in the formalism LP^{MLN} [16], and an HC is defined by an LP^{MLN} program consisting of a set of context constraints, called a *context encoding*, and a *classifier assignment encoding* expressing the probability distributions over labels for each object retrieved from a local classifier. We have shown that solutions of the resulting *HC encoding* can be obtained efficiently by means of a backtranslation from LP^{MLN} into classical answer set programs with weak constraints [1], and by leveraging combinatorial optimization capabilities of state-of-the-art ASP solvers.

Experimental results regarding object classification in indoor and outdoor scenes exhibit significant accuracy improvements compared to using only a local classifier.

5 Planned Research and Expected Outcomes

In this final section, I describe ongoing and planned work towards my main goal of improving the efficiency and usefulness of HEX in practice. It addresses open issues as well as new research questions that have emerged from my previous research work.

Evaluation Heuristics and Nogood Minimization. Further questions emerged from our work on integrating the solving process and the evaluation of external sources more closely, based on partial assignments. I plan to investigate whether the tradeoff between information gained from early calls and the additional runtime can be balanced better by employing more sophisticated evaluation heuristics. So far, only simple heuristics have been considered (i.e. external evaluation after every guess of the solver or only after every tenth guess). There are several possible heuristics for deciding the frequency of external evaluations, e.g., based on the structure of the input program. In particular, I am planning to develop novel dynamic evaluation heuristics, that decide whether an external source should be queried or not based on the information gain of previous calls.

Moreover, a related goal is to develop other more sophisticated algorithms for nogood minimization that potentially perform better than our current algorithm. Our first experiments with a simple minimization algorithm have already shown good results, and minimizing nogoods learned after a complete model candidate has been computed, showed to be the best setting in [8]. More sophisticated algorithms for conflict minimization can be found in the literature, e.g. the QUICKXPLAIN algorithm [15], which is more suited for nogoods that contain many irrelevant literals. Moreover, dynamic heuristics could also be used for deciding how many minimal nogoods to compute based on a given input-output nogood (in general, there are exponentially many).

Performance Improvements Based on Atom Dependencies. Information about dependencies over external atoms is important in most parts of the HEX algorithm. The fact that information on which part of the external atom input a given output actually depends is hidden in the black-box approach, e.g., concerns the minimality check of the HEX algorithm since it could often be skipped if this information was available. So far, a dependency graph representing an overestimation of the real dependencies is used to decide if the expensive check can be omitted. Thus, having more fine-grained information about the actual dependencies among atoms is crucial for applications where otherwise the overestimation makes the minimality check infeasible.

For pruning the dependency graph for the minimality check, there are two possible leverage points. First, the graph computed for a program in the beginning could be updated dynamically based on input assignments contained in a candidate answer set. This requires a new data structure for representing the graph allowing efficient updates. Second, often a user has more knowledge about on which part of the external atom input a given output depends, and I am planning to develop a formalism that allows the user to specify these dependencies conveniently. The additional information is likely to be also useful for grounding of external atoms and their evaluation during search.

Using External Atoms for Probabilistic Reasoning. There are at least two different options regarding the integration of probabilistic reasoning in ASP by utilizing HEX, which I am planning to investigate. First, since efficient probabilistic reasoners, e.g. *Alchemy* for *Markov Logic*, exist, these could be interfaced directly by external atoms. Then, part of a HEX program could be used to define a *Markov Network*, which the external atom receives as input. In another part of the program, the external atom could be queried (e.g. in form of conditional probability queries). A concrete use case would be an agent that executes a plan in an environment containing objects, which the agent needs to classify on the basis of their features and the agent's background knowledge.

The second option consists in leveraging existing solving techniques of ASP and HEX for computing probabilistic consequences. In the context of collective classification, the probabilities for label assignments could be derived based on other assigned labels in a program, which are in turn aggregated by an external atom that returns the most likely label for an object. Given cyclic dependencies between label assignments, collective classification corresponds to finding a *stable* label assignment.

References

1. Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Trans. Knowl. Data Eng.*, 12(5):845–860, 2000.
2. Francesco Calimeri, Susanna Cozza, and Giovambattista Ianni. External sources of knowledge and value invention in logic programming. *Ann. Math. Artif. Intell.*, 50(3-4):333–361, 2007.
3. Thomas Eiter, Michael Fink, Giovambattista Ianni, Thomas Krennwallner, Christoph Redl, and Peter Schüller. A model building framework for answer set programming with external computations. *TPLP*, 16(04):418–464, 2016.
4. Thomas Eiter, Michael Fink, and Daria Stepanova. Data repair of inconsistent nonmonotonic description logic programs. *Artif. Intell.*, 239:7–53, 2016.
5. Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. DLVHEX: A prover for semantic-web reasoning under the answer-set semantics. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2006), 18-22 December 2006, Hong Kong, China*, pages 1073–1074. IEEE Computer Society, 2006.
6. Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. Towards efficient evaluation of hex programs. In Jürgen Dix and Anthony Hunter, editors, *Proceedings of the 11th Workshop on Nonmonotonic Reasoning*, pages 40–46, Ifl-06-04, 2006. Clausthal University of Technology. 11th International Workshop on Nonmonotonic Reasoning.
7. Thomas Eiter and Tobias Kaminski. Exploiting contextual knowledge for hybrid classification of visual objects. In *JELIA*, volume 10021 of *LNCS*, pages 223–239, 2016.
8. Thomas Eiter, Tobias Kaminski, Christoph Redl, and Antonius Weinzierl. Exploiting partial assignments for efficient evaluation of answer set programs with external source access. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, pages 1058–1065. IJCAI/AAAI Press, 2016.
9. Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In José Júlio Alferes and João Alexandre Leite, editors, *JELIA 2004*, volume 3229 of *LNCS*, pages 200–212. Springer, 2004.
10. Michael Fink, Stefano Germano, Giovambattista Ianni, Christoph Redl, and Peter Schüller. ActHEX: Implementing HEX programs with action atoms. In Pedro Cabalar and Tran Cao Son, editors, *LPNMR 2013*, volume 8148 of *LNCS*, pages 317–322. Springer, 2013.
11. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In *ICLP (Technical Communications)*, volume 52 of *OASICS*, pages 2:1–2:15. Schloss Dagstuhl, 2016.
12. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.
13. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
14. Lise Getoor. *Introduction to statistical relational learning*. MIT press, 2007.
15. Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In Deborah L. McGuinness and George Ferguson, editors, *Innovative Applications of Artificial Intelligence, IAAI 2004*, pages 167–172. AAAI Press / The MIT Press, 2004.
16. Joohyung Lee and Yi Wang. Weighted rules under the stable model semantics. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning, KR 2016*, pages 145–154. AAAI Press, 2016.
17. Max Ostrowski and Torsten Schaub. ASP modulo CSP: the clingcon system. *TPLP*, 12(4-5):485–503, 2012.
18. Christoph Redl, Thomas Eiter, and Thomas Krennwallner. Declarative belief set merging using merging plans. In Ricardo Rocha and John Launchbury, editors, *Practical Aspects of Declarative Languages, PADL 2011*, volume 6539 of *LNCS*, pages 99–114. Springer, 2011.

Discovering and Proving Invariants in Answer Set Programming and Planning

Patrick Lühne

University of Potsdam, Germany,
`patrick.luehne@cs.uni-potsdam.de`

Abstract. Answer set programming (ASP) and planning are two widely used paradigms for solving logic programs with declarative programming. In both cases, the quality of the input programs has a major influence on the quality and performance of the solving or planning process. Hence, programmers need to understand how to make their programs efficient and still correct. In my PhD studies, I explore how input programs can be improved and verified automatically as a means to support programmers. One of my research directions consists in discovering invariants in planning programs without human support, which I implemented in a system called *ginkgo*. Studying dynamic systems in greater depth, I then developed *plasp* 3 with members of my research group, which is a significant step forward in effective planning in ASP. As a second research direction, I am concerned with automating the verification of ASP programs against formal specifications. For this joint work with Lifschitzs group at the University of Texas, I currently develop a verification system called *anthem*. In my future PhD studies, I will extend my research concerning the discovery and verification of ASP and planning problems.

1 Introduction

Answer set programming (ASP) and planning are two widely used paradigms for solving logic programs with declarative programming. While ASP aims to be as general-purpose as possible, planning focuses on dynamic systems, where sequences of actions are searched in order to achieve specific goals. As with other knowledge representation paradigms, quality and performance in solving and planning not only depend on the implementations of solvers and planners but also on the quality of the input program specifications. From the perspective of programmers working with solvers and planners, it is, hence, fundamentally important to understand whether their programs are efficient and, more importantly, correct with respect to their specifications.

In my PhD studies, I explore how input programs can be improved and verified automatically as a means to support programmers. Both of these objectives relate to *invariants* of logic programs—properties that preserve the correctness of a program.

My first research direction consists in *discovering* invariants without human support. For this purpose, I developed the system *ginkgo*, which continuously discovers invariants in planning problems by generalizing conflict constraints learned by an ASP solver (Section 2). While working with planning problems in the *planning domain definition language* (PDDL [8]), I further implemented *plasp* 3, the third generation of an ASP planning system. With *plasp*, PDDL programs can be solved with established ASP solvers such as *clingo*. Based on this effort, members of our research group and I showed how to make planning in ASP more effective with parallel planning (Section 3).

As a second research direction, I investigate how to automate the *verification* of ASP programs against formal specifications in a subset of the input language of *clingo* in cooperation with Vladimir Lifschitzs group at the University of Texas. *anthem*, another system that I currently develop, aims to perform this task by translating ASP programs to first-order logic formulas to validate them against a specification by a theorem prover (Section 4).

In future work, I will extend my research concerning the discovery and verification of ASP and planning problems. Section 5 discusses such directions, before Section 6 concludes this extended abstract.

2 *ginkgo*Discovering Invariants in ASP Planning

Conflict learning has become a base technology in Boolean constraint solving, and, in particular, answer set programming. However, learned constraints are only valid for a currently solved problem instance and do not carry over to similar instances. To address this issue, I developed a framework featuring an integrated feedback loop that allows for reusing conflict constraints [3]. The idea is to extract (propositional) conflict constraints, generalize and validate them, and reuse them as integrity constraints. In this way, an input program is continuously extended with automatically discovered invariants. Although I explored this approach in the context of dynamic systems (specifically, PDDL planning), the ultimate objective is to overcome the issue that learned knowledge is bound to specific problem instances.

I implemented this workflow in two systems, namely, a variant of the ASP solver *clasp* that extracts integrity constraints, along with the downstream system *ginkgo*[1] for generalizing and validating them. *ginkgo* finds invariants by first deriving candidate properties (learned constraints that are generalized over the temporal domain). These properties are then checked for invariance. This relies on automated proofs that I fully implemented in ASP with meta encodings.

3 *plasp* 3Towards Effective ASP Planning

Emerging from my work with ASP-based planning in the *ginkgo* system, I implemented the third installment of *plasp*[2]. While earlier versions of *plasp* were pure PDDL-to-ASP translators [4], *plasp* 3 was conceived to provide a flexible

[1] https://github.com/potassco/ginkgo
[2] https://github.com/potassco/plasp

platform to experiment with a variety of techniques to make planning in ASP more effective (to be published at LPNMR 2017 [2]).

For this purpose, I reimplemented *plasp*, while widening the range of accepted PDDL features in comparison to the previous versions. Further, our research group developed novel planning encodings, some inspired by SAT planning and others exploiting ASP features such as well-foundedness. I designed *plasp* 3 such that it handles multivalued fluents and, hence, captures both PDDL as well as SAS planning formats. Third, enabled by multishot ASP solving, advanced planning algorithms are offered, also borrowed from SAT planning. Empirical analyses show that these techniques have a significant impact on the performance of ASP planning.

4 *anthem*Verifying the Correctness of ASP Programs

In their recent work (to be published [5,6]), Harrison, Lifschitz, and Raju have extended the definition of program completion to a subset of the input language of *clingo*. The aim of their work is to extend the applicability of formal verification methods to ASP by turning logic programs into completed definitions. This can also be understood as a translation from *clingo*s input language to first-order logic formulas.

In cooperation with their research group at the University of Texas, I started developing a system called *anthem*,[3] which performs the completion of logic programs automatically. After translating and simplifying formulas with *anthem*, programmers can see more clearly what exactly their program solves.

Furthermore, the first-order logic representation allows us to perform automated proofs by employing established theorem provers, which commonly operate on similar input formats. Popular theorem provers include *E* [10], *Coq* [1], and *Prover9* [7]. Our goal is to extend *anthem* such that it can be used to quickly test whether ASP programs fulfill given invariants. With such a tool, programmers could start writing programs by first making a formal specification, against which their code is later verified.

5 Future Work

As stated before, my most recent work focuses on using theorem provers to verify that logic programs comply with a given specification. This indirection of proving invariants through first-order logic might turn out particularly useful when coming back to my earlier research on the *ginkgo* system. This is because there are many established first-order theorem provers, which might make for a stronger proof system than the counterexample-based validation method currently employed by *ginkgo*.

Furthermore, I will expand my research in the field of PDDL planning. Building on *plasp* 3, I want to explore how far planning in ASP can be pushed, with

[3] https://github.com/potassco/anthem

the objective of achieving performance on par with state-of-the-art SAT planners such as *Madagascar* [9].

Finally, I am always exploring opportunities to apply the planning-related techniques to the broader scope of general ASP programs. This involves the research areas of automatic modeling, program synthesis, and superoptimization, which I want to further familiarize myself in my upcoming PhD studies.

6 Conclusions

Concerning the automatic discovery and verification of invariants in logic programs, I already made insightful progress. In my early PhD projects, I showed the feasibility of reusing learned conflict constraints in ASP planning by means of generalization with my *ginkgo* system. I further studied dynamic systems as such and helped making ASP planning much more effective with *plasp* 3 and performance-wise closer to state-of-the-art SAT planners than previous attempts. I believe that these two systems I developed could make use of refined invariant finding techniques, which is one of the things I want to study in the remainder of my PhD studies.

Furthermore, I am researching automated verification techniques in multiple contexts. First, as a means to validate potential candidate invariants within *ginkgo*. Second, to completely automate the process of testing ASP programs against formal specifications, which is the objective of my currently work-in-progress system *anthem*. This is a technique that could later be useful for other parts of my research as well.

To my mind, there are many interesting aspects of discovering and verifying invariants ahead that I want to address in my PhD studies. This also includes more practical applications such as making ASP planning yet more effective.

References

1. Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. The Coq proof assistant reference manual: Version 8.6. https://coq.inria.fr/distrib/current/refman/, 2016.
2. Y. Dimopoulos, M. Gebser, P. Lhne, J. Romero, and T. Schaub. *plasp* 3: Towards effective ASP planning *(to appear)*. In *Proceedings of the Fourteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2017.
3. M. Gebser, R. Kaminski, B. Kaufmann, P. Lhne, J. Romero, and T. Schaub. Answer set solving with generalized learned constraints. In M. Carro and A. King, editors, *Technical Communications of the Thirty-second International Conference on Logic Programming (ICLP'16)*, volume 52, pages 9:1–9:15. Open Access Series in Informatics (OASIcs), 2016.
4. M. Gebser, R. Kaminski, M. Knecht, and T. Schaub. plasp: A prototype for PDDL-based planning in ASP. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic*

Reasoning (LPNMR'11), volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 358–363. Springer-Verlag, 2011.
5. Amelia Harrison, Vladimir Lifschitz, and Dhananjay Raju. Program completion in the input language of gringo. Submitted for publication, 2017.
6. Vladimir Lifschitz. Achievements in answer set programming (preliminary report). In *Working Notes of the Workshop on Answer Set Programming and Other Computing Paradigms*, 2016.
7. W. McCune. Prover9 and Mace4. http://www.cs.unm.edu/~mccune/prover9/, 2005–2010.
8. D. McDermott. PDDL—the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
9. J. Rintanen. Madagascar: Scalable planning with SAT. In M. Vallati, L. Chrpa, and T. McCluskey, editors, *Proceedings of the Eighth International Planning Competition (IPC'14)*, pages 66–70. University of Huddersfield, 2014.
10. Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

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. In *European Journal of Operational Research*, volume 3, pages 283–288. Elsevier Science, 1981.

Extensions of Answer Set Programming: Declarative Heuristics, Preferences and Online Planning

Javier Romero

University of Potsdam, Germany

Abstract. The goal of this thesis is to extend Answer Set Programming (ASP) with declarative heuristics, preferences, and online planning capabilities. For declarative heuristics, the thesis presents a general declarative approach for incorporating domain-specific heuristics into ASP solving by means of logic programming rules. For preferences, the approach developed in my thesis and the resulting `asprin` system provide a general and flexible framework for quantitative and qualitative preferences in ASP. For online planning, the goal of my thesis is to integrate different approaches to online planning with incomplete information in a unified ASP approach.

1 Introduction

Answer Set Programming (ASP; [1]) is a well established approach to *declarative problem solving*. Rather than solving a problem by telling a computer *how to solve the problem*, the idea is to simply describe *what the problem is* and leave its solution to the computer. The success of ASP is due to the combination of a rich yet simple modeling language with high-performance solving capacities. Modeling has its roots in the fields of Knowledge Representation and Logic Programming, while solving is based in methods from Deductive Databases and Satisfiability Testing (SAT; [2]). ASP programs resemble Prolog programs, but they are interpreted according to the stable models semantics [12], and the underlying solving techniques are closely related to those of modern SAT solvers. The goal of my doctorate is to develop different extensions of ASP: for declarative heuristics, preferences, and online planning.

2 Declarative Heuristics

From the solving perspective, for solving real-world problems in ASP, it is sometimes advantageous to take an application-oriented approach by including domain-specific information. On the one hand, domain-specific knowledge can be added for improving deterministic assignments through propagation. On the other hand, domain-specific heuristics can be used for making better nondeterministic assignments. To this end, in my thesis I introduce a general declarative framework for incorporating domain-specific heuristics into ASP solving

[11], using a directive `#heuristic` whose arguments allow us to express various modifications to the solver's heuristic treatment of atoms. The directives are interpreted as a new type of rules, that are subsequently exploited by the solver when it comes to choosing an atom for a non-deterministic truth assignment. The heuristic framework offers completely new possibilities of applying, experimenting, and studying domain-specific heuristics in a uniform setting. In the current stage, heuristic directives are an integral part of `clingo5` and have already been used in some real world applications. The next step is to extend the approach using machine learning techniques for automatically learning heuristic rules.

3 Preferences

Another extension that is often necessary in real-world applications is being able to represent and reason about preferences. This was realized quite early in ASP, leading to many approaches to preferences [7, 4, 14]. Departing from there, the approach developed in my thesis [5, 6] and the resulting `asprin`[1] system provide a general and flexible framework for quantitative and qualitative preferences in ASP. This framework is *general* and captures many of the existing approaches to preferences. It is *flexible*, providing means for the combination of different types of preferences. And it is also *extensible*, allowing for an easy implementation of new approaches to preferences. The next steps for this part of my thesis are to finish the implementation of a stable and user-friendly version of the system, to implement new types of preferences in the system (f.e., CP-nets), and to integrate unsatisfiable-core solving techniques into the approach.

4 Online Planning

The third part of my thesis is focused on extensions for online planning with ASP. Planning is one of the earlier applications of ASP [13]. It has been extended to deal with incomplete information about the initial state and/or sensing actions, and for solving conformant [10, 15, 17] and conditional planning [16] problems. Alternative approaches outside ASP for dealing with incomplete information and sensing actions include *assumption-based planning* [8] and *continual planning* [3]. The goal of this last part of the thesis is to integrate these techniques in a unified ASP framework, and apply it to a real-world application in robotics.

References

1. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.

[1] `asprin` stands for "*AS*P for *pr*eference handl*in*g".

2. 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.
3. M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, 2009.
4. G. Brewka. Complex preferences for answer set optimization. In D. Dubois, C. Welty, and M. Williams, editors, *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, pages 213–223. AAAI Press, 2004.
5. G. Brewka, J. Delgrande, J. Romero, and T. Schaub. asprin: Customizing answer set preferences without a headache. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-Ninth National Conference on Artificial Intelligence (AAAI'15)*, pages 1467–1474. AAAI Press, 2015.
6. G. Brewka, J. Delgrande, J. Romero, and T. Schaub. Implementing preferences with asprin. In F. Calimeri, G. Ianni, and M. Truszczyński, editors, *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15)*, volume 9345 of *Lecture Notes in Artificial Intelligence*, pages 158–172. Springer-Verlag, 2015.
7. G. Brewka, I. Niemel, and M. Truszczyński. Answer set optimization. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 867–872. Morgan Kaufmann Publishers, 2003.
8. S. Davis-Mendelow, J. Baier, and S. McIlraith. Assumption-based planning: Generating plans and explanations under incomplete knowledge. In desJardins and Littman [9], pages 209–216.
9. M. desJardins and M. Littman, editors. *Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence (AAAI'13)*. AAAI Press, 2013.
10. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning. *Artificial Intelligence*, 144(1-2):157–211, 2003.
11. M. Gebser, B. Kaufmann, R. Otero, J. Romero, T. Schaub, and P. Wanko. Domain-specific heuristics in answer set programming. In desJardins and Littman [9], pages 350–356.
12. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.
13. V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.
14. T. Son and E. Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 6(5):559–608, 2006.
15. T. Son, P. Tu, and C. Baral. Planning with sensing actions and incomplete information using logic programming. In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Artificial Intelligence*, pages 261–274. Springer-Verlag, 2004.
16. P. Tu, T. Son, and C. Baral. Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. *Theory and Practice of Logic Programming*, 7:377–450, 2007.
17. P. Tu, T. Son, M. Gelfond, and A. Morales. Approximation of action theories and its application to conformant planning. *Artificial Intelligence*, 175(1):79–119, 2011.

Theory Reasoning with Answer Set Programming

Sebastian Schellhorn[1]

[1]University of Potsdam, Germany

Abstract. Answer Set Programming (ASP) is a well known declarative solving paradigm to model and solve efficiently combinatorial real world problems. But it may suffer from grounding bottleneck by represent and treat elements of other theories (eg linear constraints) or variables over multivalued domains. Addressed to this problem, my research focus on ASP modulo theory reasoning. The recent available system *clingo* 5 provides a generic interface to enhance ASP with theory reasoning capabilities. I instantiate this framework with linear constraints and elaborate upon its formal properties. My goal is to extend the picture ASP modulo theories and its relations. Furthermore, my research interests focus on extensions of stable model semantics regarding foundedness regarding multivalued domains. Finally, I show some state of the art approaches addressed to these problems, main issues and ideas to tackle them.

1 Introduction

Answer Set Programming (ASP[6]) is a well known declarative problem solving paradigm to efficiently solve combinatorial problems. It features a simple and flexible modeling language. Thus with ASP it is possible to describe real world problems in an intuitive way. State-of-the-art ASP solvers use high performance solving technologies and are able to efficiently solve several of these real world problems. However, certain problems require a more natural modeling with constraints over reals or integers. In this case, ASP solvers suffer from the grounding bottleneck by represent and treat elements of other theories invoke non-Boolean constraints (eg linear constraints) or variables over, probably infinite, multivalued domains. Addressed to this problem, my research focus on ASP modulo theory reasoning.

The recent available system *clingo* 5 [3] provides a generic interface to enhance ASP with theory reasoning capabilities. I instantiate this framework with Linear Programming (LP[1]) constraints and elaborate upon its formal properties. My goal is to extend and generalize this elaboration on formal relations of ASP with different theories, solve corresponding issues and extend the picture of ASP semantics and theory reasoning.

Furthermore, my research interests focus on extensions of stable model semantics regarding foundedness over multivalued domains. In particular, I am interested in stable model semantics extensions based on logic of Here-and-There

(HT[11]) regarding to default values of multivalued domain variables, foundedness, partial functions and aggregates. Thus I plan to achieve a Bound Founded ASP (BFASP[5]) semantics for arbitrary ordered multivalued variable domains.

Finally, in the following I show some state-of-the-art approaches addressed to these problems, clarify main issues and present ideas to tackle them.

2 ASP modulo Linear Constraints

In the past there were several approaches to combine ASP with constraints starting with [12]. Usual approaches are satisfiability checks on full assignments by using dedicated solver as a black box or adapting the underlying ASP solving algorithms to achieve an extension to a desired theory. With the recent *clingo* 5 interface, following the approach of lazy theory solving[4], it is possible to extend ASP with theories in a much easier way than before. This interface offers the opportunity to come up with an own propagator for a specific theory and use it during search on partial assignments to derive new information given by the theory or rule out search trees which are inconsistent with the theory.

I first started to study linear constraints over reals and its possible treatments, like translation based approaches, which still suffer from the grounding bottleneck. Afterward, I combined ASP with linear constraints over reals and integers on a system and theoretical level using the mentioned interface. Thereby, I abstract from the specific semantics of the theory by considering the linear constraint atoms (lc-atoms) associated with its constraints, and deal with the constraints as usual in LP. My *clingo* derivative *clingo*[LP], uses an easy modifiable Python script, providing a simple propagator. The propagator follows a state based approach. To solve the sub problem given by a set of linear constraints *clingo*[LP] supports LP solvers *cplex* and *lpsolve* as black boxes, which are plugged by its Python interface. Thus, to the best of my knowledge, it is the first time that Mixed Integer Linear Programming (MILP) solvers are combined with ASP without using a translation based approach for example like *mingo* [13] does. *clingo*[LP] checks during search if a corresponding set of linear constraints is satisfiable. Mention that it is the first implementation of *clingo* with linear constraints over reals. Furthermore, it achieves superior results on metabolic network completion of *Escherichia coli*[10], by using a new hybrid approach that integrates a previous qualitative ASP approach with quantitative means of a linear constraint set.

On the theoretical side, there are in general four possibilities to interpret lc-atoms. I analyzed them regarding their lc-stable models and compared them to each other and regular stable models to elaborate their formal relations. I figured out that most of them can be ordered independent of the corresponding theory, except for one proposition. Furthermore, I found translations to represent the two incomparable but most intuitive possibilities in sense of ASP by each other regarding to programs with specific properties.

Finally, with *clingo*[LP] I finished one practical and theoretical part of my research. An open issue would be to generalize and elaborate these propositions and translations to less restrictive conditions and other theories. Additionally in a long run, I want to extend *clingo*[LP] by a kind of domain propagation over reals to avoid the intermediate search regarding to each partial assignment given by the black box approach.

3 ASP Foundedness over Multivalued Domains

On the other hand, my research focuses on derivatives of ASP semantics combined with default values over multivalued domains, foundedness, partial functions and aggregates.

The foundedness idea of BFASP represented in [5] is to derive a maximal value of a multivalued domain variable where we have a reason for. To achieve foundedness for variables with arbitrary ordered multivalued domains it is necessary to introduce a monotonic operator to declare reasonable values for assignments. Since the presented semantics of BFASP lacks the intuition of ASP in some cases, there is the need to take up the idea and reinvent it in different settings like logic of Here-and-There.

So far, there exists several ideas to achieve default valuations for multivalued domains like the approach of extending the logic of Here-and-There wich constraints [8], which allows default ranges for assignments. But this formalism itself is not able to grasp the intuition of BFASP, since their assignment operator uses an anti-monotonic relation. Furthermore, this approach misses an order on assignments to express preferred valuations.

Thus, I plan to come up with new stable model semantics for linear constraint variables with arbitrary ordered multivalued domains closely related to BFASP and logic of Here-and-There with constraints. To this end, I figured out two possible semantics in the context of logic of Here-and-There to achieve this idea of BFASP, which are closely related to ASP semantics.

While thinking about new semantics, I recognize similar problems as we can observe with aggregates and partial functions regarding to monotonic and anti-monotonic behaviors. Thus, I want to find relations of my semantics to aggregates and partial functions to extend the picture of their treatments and investigate reasons for counter intuitive behaviors as well. To this end, I reformulated different aggregate semantics in context of logic of Here-and-There to set the same footing for my comparison. Finally, I want to figure out the strength and weakness of my semantics compared to well known approaches.

However, in my main research tasks I focused on linear constraints over real-valued variables, but it is also possible to think about arbitrary functions over ordered multivalued domains and their monotonic behaviors to extend this approach.

References

1. G. Dantzig. *Linear Programming and Extensions.* Princeton, 1963.

2. M. Carro and A. King, editors. *Technical Communications of ICLP'16*, OASIcs, 2016.
3. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and P. Wanko. Theory solving made easy with clingo 5. In [2], 2:1-2:15.
4. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, 26:825-885. IOS, 2009.
5. R. Aziz. Bound Founded Answer Set Programming. In CoRR, abs/1405.3367, 2014.
6. M. Gebser, R. Kaminski, B. Kaufmann and T. Schaub. *Answer Set Solving in Practice*, Morgan and Claypool Publishers, 2012.
7. R. Kambhampati, editor. *Proceedings of IJCAI'16*, IJCAI/AAAI Press, 2016.
8. P. Cabalar, R. Kaminski, M. Ostrowski and T. Schaub. An ASP Semantics for Default Reasoning with Constraints. In [7], 1015-1021, 2016.
9. M. Balduccini and T. Janhunen, editors. *Proceedings of LPNMR'17*, Springer, 2017.
10. C. Frioux and T. Schaub and S. Schellhorn and A. Siegel and P. Wanko. Hybrid Metabolic Network Completion. In [9], to appear, 2017.
11. D. Pearce. A new logical characterisation of stable models and answer sets. In *Proceedings of the Sixth International Workshop on Non-Monotonic Extensions of Logic Programming (NMELP'96)*, pages 57–70. Springer-Verlag, 1997.
12. S. Baselice, P. Bonatti and M. Gelfond. Towards an integration of answer set and constraint solving. In *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, M. Gabbrielli and G. Gupta, Eds. Lecture Notes in Computer Science, vol. 3668. Springer-Verlag, 52–66, 2005.
13. G. Liu, T. Janhunen, and I. Niemelä. Answer set programming via mixed integer programming. In *Proceedings of KR'12*, 32-42. AAAI, 2012.

Lazy Grounding and Heuristic Solving in Answer Set Programming

Richard Taupe[1,2]

[1] Alpen-Adria-Universität, Klagenfurt, Austria,
rtaupe@edu.aau.at
[2] Siemens AG Österreich, Corporate Technology, Vienna, Austria,
richard.taupe@siemens.com

Abstract. Most ASP systems suffer from the so-called *grounding bottleneck*, i.e. they cannot solve problems whose propositional grounding exceeds given memory limits. Lazy-grounding solvers mitigate this issue by interleaving grounding with search but they cannot compete with state-of-the-art solvers in terms of runtime performance. The goal of this dissertation is to enhance the power of lazy-grounding solvers by exploring conflict-driven nogood learning and various forms of heuristics.

Keywords: answer set programming, lazy grounding, heuristics

1 Introduction

Answer Set Programming (ASP) is an approach to declarative problem solving [2, 14]. In this approach, problems to be solved by a computer are encoded as logic programs, which are sets of rules that can contain quantified variables. Most ASP systems split the solving process into two steps: First, a *grounder* transforms the input program containing variables into a propositional encoding. Then, solutions for the resulting variable-free program are found by a *solver*. The grounding step can result in an exponential blow-up in space [3, 8, 9, 11].

This so-called *grounding bottleneck* is a major problem of traditional approaches to answer set programming (ASP), which follow the *ground-and-solve* paradigm. For example, the constraint

$$\leftarrow\ sp(SP, P1),\ sp(SP, P2),\ P1 \neq P2.$$

has to be replaced by up to $|SP| \cdot |P1| \cdot |P2|$ ground constraints, where $|SP|$, $|P1|$ and $|P2|$ are the number of values the respective variable may take. Many of these ground constraints may be irrelevant because they cannot be satisfied in a given problem instance anyway. Problems that are actually easy to solve thus become prohibitive as soon as their grounding does not fit into working memory. This makes ASP, an otherwise powerful and versatile approach, unsuitable for large-scale problem instances frequently occurring in real-world settings.

State-of-the-art grounders like GRINGO [9] and $\mathcal{I}$-DLV [3, 8] employ sophisticated grounding techniques to omit irrelevant ground rules, but these can only reduce and not eliminate the exponential blow-up in the worst case. *Lazy grounding* interleaves grounding and solving to avoid storing the entire ground program in memory. Known approaches to lazy grounding are ASPeRiX [13], GASP [5], OMiGA [6, 16], and, most recently, Alpha [17]. Lazy grounding methods have also been proposed for FO(·), a knowledge representation formalism whose foundations are similar to those of ASP [4].

While lazy-grounding systems are able to limit their memory usage, their time consumption is not comparable to that of state-of-the-art solvers. One reason for this is that most of these systems do not exploit conflict-driven nogood learning (CDNL), which is a key success factor of state-of-the-art ASP solvers. Alpha has been the first lazy-grounding system to employ CDNL [17]. It consists of a grounder and a solver which, however, do not work in sequence (as in ground-and-solve), but interact cyclically. Alpha does not, however, reach the performance of traditional solvers yet. One reason for this is that Alpha (and all other lazy-grounding systems) lack powerful search heuristics to guide the exploration of the search space, which are another major success factor of traditional systems.

2 Research Questions

This leads to the central research questions of this thesis:

1. How can lazy-grounding solvers be enabled to solve large-scale (industrial) problem instances as efficiently as traditional solvers solve smaller instances?
2. How can conflict learning contribute to that goal, and can conflicts be reused across problem instances?
3. How can various forms of heuristics, e.g. domain-independent or domain-specific search heuristics, contribute to that goal?

Lazy grounding methods will be evaluated on real-world industrial problem instances from domains like cyber-physical (production) systems, road traffic control, and railway operation.

3 Accomplished Work

Source code has been contributed to Alpha, a new lazy-grounding system introduced by Antonius Weinzierl [17]. Contributions are made under an open-source license and are freely available at `https://github.com/alpha-asp`.

While initial work aimed at comprehending the solver's inner workings and making small improvements and extensions on the go, our focus has soon shifted to the development of domain-independent search heuristics.

3.1 Heuristics for Lazy-Grounding ASP Solving

Alpha takes ideas from state-of-the-art pre-grounding ASP solvers. Therefore it is natural to try and adopt heuristics from such systems as well. Ongoing work has focused on heuristics for the *solver* component, while heuristics for the *grounder* in a lazy-grounding system are subject of future work. Heuristics for answer-set solving can roughly be classified as follows: *domain-independent heuristics* do not take the nature of the problem at hand into account, whereas *domain-specific heuristics* have to be tailored to a specific problem. For the class of domain-independent heuristics, a prominent example is *BerkMin* [12], which is a domain-independent heuristic originally developed for SAT but also successfully employed by ASP solvers (such as CLASP [11] and WASP [1]). It organizes the set of conflict clauses as a chronologically ordered stack, thereby preferring variables in recent conflicts. This is done to regard the fact that the set of variables responsible for conflicts may change very quickly. Additionally, a so-called *activity* counter is assigned to every variable that counts the number of clauses involving this variable that are responsible for at least one conflict. These counters are divided by a constant ("decayed") periodically to reduce the influence of "aged" clauses. When BerkMin is asked for an atom, it chooses the most active unassigned atom in the nogood nearest to the top of the stack that is not yet fully assigned. Other counters are maintained for picking truth values.

A direct application of *BerkMin* to a lazy-grounding ASP solver like Alpha seems unnatural because such a solver differs in many important ways from a solver adhering to the classical ground-and-solve paradigm. One major difference is that not all ground rules, and consequently not all ground literals and atoms, are known at any time to a lazy-grounding solver. Because of this, a *BerkMin*-like heuristic applied to lazy grounding can only incorporate atoms that are already known to the solver. Another major difference lies in the solving mechanism: while a traditional ASP solver can choose any atom to guess on, Alpha only guesses on atoms representing bodies of applicable rules, i.e. on rules whose positive body holds and whose negative body is not contradicted in the current assigment. In other words, Alpha only guesses whether an applicable rule fires, but it does not pick truth values for specific atoms.

3.2 Domain-Independent Heuristics in Alpha

Branching heuristics choose both an atom and a truth value to assign to this atom. If the solver discovers an inconsistency after performing this assignment, Alpha backtracks and tries to assign the inverse truth value to the same atom. The following paragraphs contain brief descriptions of the branching heuristics developed so far. Details can be found in [15].

A Naive Heuristic (N). The naive heuristic implemented in Alpha works as follows: Active choice points are organized as a FIFO queue, i.e. the earlier a choice point is grounded the earlier it is chosen to be guessed on. The truth value is always chosen to be true.

A BerkMin-Inspired Heuristic (BMI). An initial application of BerkMin to Alpha looks as follows: To choose an atom, iterate through the stack of nogoods until one is encountered that contains literals that are not yet assigned, which are then transformed into atoms (by removing their sign). From these atoms, the most active choice point is then chosen (i.e., the atom with the highest activity counter that represents the body of an applicable rule). To choose the truth value, a slightly modified version of BerkMin's procedure has been implemented that incorporates the *must-be-true* truth value only present in lazy-grounding systems by picking true if the atom is already must-be-true (i.e. it has to be true in a solution but has not yet been derived).

BMI with a Bounded Queue of Choice Points (BBQ). *BMI* is still inefficient because it maintains a stack of nogoods even though only a small subset of the atoms in these nogoods are considered as valid choice points. In fact, choosing the most active choice point in a nogood is mostly useless because most if not all nogoods will contain only a single literal representing a rule body. Therefore, *BBQ* manages a bounded FIFO queue of choice points occurring in recent nogoods. When new nogoods arrive (by grounding or learning), the oldest choice points leave the queue. The branching heuristic then chooses the most active atom from this queue, disregarding the order in which they appear.

A Dependency-Driven Heuristic (DD). The BerkMin variants *BMI* and *BBQ* described so far suffer from the fact that choice points comprise only a small portion of all the literals occuring in nogoods and therefore do not influence activity and sign counters as much as other atoms. The basic idea of *DD* is therefore to find *dependent* atoms that can enrich the information available for a choice point. *DD* manages a stack of nogoods similar as *BMI* and starts by looking at the most active atom a in the nogood currently at the top of the stack. If a is an active choice point (i.e. representing the body of an applicable rule), it is immediately chosen; else the most active choice point dependent on a is. If there is no such atom, *DD* continues further down the stack.

A Generalized Dependency-Driven Heuristic (GDD). *GDD* is a more general variant of *DD* that does not consider what role an atom plays in a rule and works only with nogoods.

Pyromaniacal Variants (DD-Pyro and GDD-Pyro). One weakness of Alpha that has to be addressed in future work is its lack of support nogoods (except in one special case, cf. [17]). This means that the solver cannot recognize when an atom occurring negatively in a rule body cannot be satisfied anymore, thus exploring large portions of the search space in search for a witness. Therefore, Alpha currently benefits in many cases from heuristics that assign true to choice points whenever they can. This modification is applied to *DD* and *GDD*, which then always assign true first and try false only when backtracking. These variants are called *pyromaniacal* since they prefer the firing of a rule over its non-firing.

3.3 Evaluation

The heuristics mentioned so far are described in detail in [15], which includes the results of a basic experimental evaluation on a number of benchmark problems.

Although the heuristics presented are still under development and only a brief experimental study was conducted, promising results can be seen. The novel family of dependency-driven heuristics (*DD*, *GDD*, etc.) was repeatedly able to outperform Alpha's naive heuristic (N) as well as the two BerkMin-inspired heuristics (*BMI* and *BBQ*). The pyromaniacal variants of *DD* seem exceptionally suited to support Alpha's current decision procedure: they performed considerably above average on every occasion and beat all other heuristics more often than not. While our results are encouraging, there is obviously more work to be done to improve the performance of these heuristics and the solver in general.

4 Further Plans

Until now, our newly developed heuristics have been evaluated only in very basic experiments on a personal computer, in which the numbers of choices done by the solver have been measured. A more detailed analysis recording data like numbers of conflicts as well as time and space requirements in addition to numbers of choices is currently being conducted by means of a controlled benchmarking environment on dedicated hardware. This analysis will also comprise a larger set of benchmark problems and compare the system to other ASP solvers, especially lazy-grounding ones. Results will be published in the near future. It is also evident that the new heuristics are not yet competitive with those of state-of-the-art pre-grounding solvers. Several ideas for new variants already exist and will be tried.

While we have concentrated on domain-independent search heuristics so far, at least two other types of heuristics remain to be explored in future work: Domain-specific search heuristics, and grounding heuristics. Domain-specific heuristics have been proposed for pre-grounding solvers (cf. HCLASP [10] and HWASP [7]) but are not directly transferable to a lazy-grounding system for the same reasons that domain-independent heuristics are not. Still, they may provide a starting point to pursue the goal of developing a uniform methodology to specify problem-specific heuristics for a lazy-grounding CDNL-based ASP solver like Alpha. Initial experiments in this direction have been conducted by Gottfried Schenner and are described in our joint paper [15]. We will design novel interfaces and extensions to the core solving algorithms to allow such problem-specific heuristics. Grounding heuristics, on the other hand, are relevant for lazy-grounding systems only. They deal with questions like how much to ground at a given point in time and when to save space by forgetting grounded nogoods or doing a restart. Some work in this direction has already been done for FO($\cdot$) [4]. Ways to adapt and extend this for use by a CDNL-based solver will be explored.

Furthermore, we plan to explore program analysis techniques (atom dependencies, strongly connected components, etc.) to guide the selection or tuning of solver heuristics. Also, our research will not be limited to heuristics but include other important issues of lazy-grounding ASP systems as well. One such issue is to extend conflict-driven nogood learning (CDNL) to learn in a non-ground way across problem instances.

References

1. Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: A native ASP solver based on constraint learning. In: LPNMR. LNCS, vol. 8148, pp. 54–66. Springer (2013)
2. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011)
3. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: $\mathcal{I}$-DLV: The new intelligent grounder of DLV. In: Adorni, G., Cagnoni, S., Gori, M., Maratea, M. (eds.) AI*IA, LNCS, vol. 10037, pp. 192–207. Springer International Publishing (2016)
4. de Cat, B., Denecker, M., Stuckey, P.J., Bruynooghe, M.: Lazy model expansion: Interleaving grounding with search. In: Journal of Artificial Intelligence Research. pp. 235–286 (2015)
5. Dal Palù, A., Dovier, A., Pontelli, E., Rossi, G.: GASP: Answer set programming with lazy grounding. Fundamenta Informaticae 96(3), 297–322 (2009)
6. Dao-Tran, M., Eiter, T., Fink, M., Weidinger, G., Weinzierl, A.: Omiga: An open minded grounding on-the-fly answer set solver. In: JELIA. LNAI, vol. 7519, pp. 480–483. Springer (2012)
7. Dodaro, C., Gasteiger, P., Leone, N., Musitsch, B., Ricca, F., Schekotihin, K.: Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). TPLP 16(5-6), 653–669 (2016)
8. Faber, W., Leone, N., Perri, S.: The intelligent grounder of DLV. In: Correct Reasoning, LNCS, vol. 7265, pp. 247–264. Springer (2012)
9. Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in gringo series 3. In: LPNMR. LNCS, vol. 6645, pp. 345–351. Springer (2011)
10. Gebser, M., Kaufmann, B., Otero, R., Romero, J., Schaub, T., Wanko, P.: Domain-specific heuristics in answer set programming. In: AAAI Conference on Artificial Intelligence. pp. 350–356. AAAI Press (2013)
11. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artificial Intelligence 187-188, 52–89 (2012)
12. Goldberg, E., Novikov, Y.: Berkmin: A fast and robust SAT-solver. In: DATE. pp. 142–149. IEEE (2002)
13. Lefèvre, C., Béatrix, C., Stéphan, I., Garcia, L.: `ASPeRiX`, a first-order forward chaining approach for answer set computing. TPLP FirstView, 1–45 (2017), `https://doi.org/10.1017/S1471068416000569`
14. Lifschitz, V.: What is answer set programming? In: AAAI Conference on Artificial Intelligence (2008), `http://www.aaai.org/Papers/AAAI/2008/AAAI08-270.pdf`
15. Taupe, R., Weinzierl, A., Schenner, G.: Introducing heuristics for lazy-grounding ASP solving. In: 1st International Workshop on Practical Aspects of Answer Set Programming (2017), to appear
16. Weinzierl, A.: Learning non-ground rules for answer-set solving. In: 2nd Workshop on Grounding and Transformations for Theories With Variables (2013)
17. Weinzierl, A.: Blending lazy-grounding and CDNL search for answer-set solving. In: LPNMR (2017), to appear, preprint available at `http://www.kr.tuwien.ac.at/research/systems/alpha/blending_lazy_grounding.pdf`

Author Index