Hybrid Compilation-based ASP solving

Carmine Dodaro, Giuseppe Mazzotta and Francesco Ricca

Department of Mathematics and Computer Science, University of Calabria, Rende, Italy

Abstract

Answer Set Programming (ASP) is a widely recognized formalism for Knowledge Representation and Reasoning. State-of-the-art ASP systems, based on the well-known *Ground&Solve* approach, are subject to the grounding bottleneck problem that, in some cases, makes the computation of answer sets practically unfeasible. Compilation-based approaches have recently demonstrated how grounding can be effectively bypassed by compiling rules into propagators, but, compiling an entire ASP program is not always advantageous. In the paper titled "Blending grounding and compilation for efficient ASP solving", presented during the "Twenty-First International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)", we proposed a hybrid approach that allows for unrestricted blending of grounding and compilation. In this paper, we investigate the advantages of using ad-hoc hybrid solvers and discuss future directions in this line of research.

Keywords

Answer Set Programming, Compilation-Based ASP Solving, Grounding Bottleneck, Hybrid Solving

1. Introduction

Answer Set Programming (ASP) [1, 2] is a well-known declarative AI formalism for KRR. Thanks to the availability of efficient implementations, ASP finds extensive applications in several AI sub-areas [3], such as Planning [4], Scheduling [5, 6], Natural Language Processing [7, 8, 9], and Databases [10, 11, 12].

Traditional ASP systems, such as CLINGO [13] and DLV [14], are based on the *Ground&Solve* approach [15]. Intuitively, an input program is first "grounded" to compute a variable-free equivalent propositional program; and subsequently, the grounded program is "solved" by employing a CDCL-like algorithm [16] that computes its answer sets. However, such an approach intrinsically suffers from the so-called *grounding bottleneck*, i.e., getting rid of variables already consumes all the computational resources (i.e., time and/or space) in several cases of practical interest [17, 18].

The grounding bottleneck [19] has been approached from several perspectives. These include hybrid formalisms [20, 13, 18, 21], *lazy grounding* architectures [22, 23, 24, 25, 26], complexity-driven program rewritings [27, 28], and program compilation into propagators [29, 30, 31, 32]. Among these, compilation-based approaches demonstrated that the grounding can be skipped in some relevant cases by compiling rules in propagators able to ground rules' inferences. Such techniques were first proposed for subprograms acting as constraints [30, 31]. More recently, the ProASP system [32] has been proposed. ProASP demonstrated that it is possible to devise a compiler also for rules "generating" answer sets (i.e., involving non-stratified negation). In ProASP, a tight [33] non-ground input program is first pre-processed by applying a rewriting encompassing program completion [34] and normalization (i.e., it produces rules of two kinds). Then, the compiler generates code for both Herbrand base generation and rule propagators. That code is injected in the CDCL solver Glucose [35] to initialize variables and simulate the presence of ground rules, respectively. In this way, the ProASP compiler produces a solver specific for the non-ground program in input that needs no grounder. Clearly, ProASP implements an approach that is basically at the antipodes of *Ground&Solve*, since it compiles all rules of the program.

However, empirical evidence showed that compiling all rules of an ASP program does not pay off in all cases w.r.t. traditional approaches [31]. This is not surprising. It is well-established that there is no free lunch in ASP solving [36], i.e., no single algorithm is the best in all cases. Under typical operational

AIxIA 2024 Discussion Papers - 23rd International Conference of the Italian Association for Artificial Intelligence, Bolzano, Italy, November 25–28, 2024

carmine.dodaro@unical.it (C. Dodaro); giuseppe.mazzotta@unical.it (G. Mazzotta); francesco.ricca@unical.it (F. Ricca)
0000-0002-5617-5286 (C. Dodaro); 0000-0003-0125-0477 (G. Mazzotta); 0000-0001-8218-3178 (F. Ricca)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-3915/Paper-5.pdf

scenarios, it is plausible that certain rules of the program can be efficiently grounded and, thus, are amenable to be processed using the traditional architecture; whereas, for the remaining part of the program, which is made of rules that are subject to the grounding bottleneck, a compilation-based system could offer a viable solution for their evaluation. This latter case, which is very common in practice, cannot be approached in the best possible way with current state-of-the-art ASP systems, that either ground or compile everything.

In the paper "Blending grounding and compilation for efficient ASP solving", we presented a compilation-based architecture, built on top of the ProASP system, for blending grounding and compilation. This novel approach allows for an unrestricted blending of both methods and aims at achieving superior performance by combining the advantages of both approaches.

In this paper, we overview hybrid ASP solving, outline its strengths, and discuss future directions of this line of research.

2. Hybrid ProASP: Blending Grounding and Compilation

In this section we describe the extended architecture of the ProASP system that exploits an optimized rewriting strategy and allows for the compilation of a program into a hybrid ASP solver (i.e. mixing *Ground&Solve* and Compilation) without any compilability restrictions that existing compilation-based approaches [30, 31] are subject to.

The ProASP system implements two different stages, namely *Compilation* and *Solving* stages. The former is used to build a hybrid solver starting from a non-ground program Π, while the latter employs the obtained ad-hoc hybrid solver to compute an answer set $\Pi \cup F$, where F is a set of facts representing a specific problem instance. At *Compilation Stage* ProASP system takes as input a non-ground program Π and compiles it into a hybrid solver. To this end, the input program Π is fed into the *Compiler* module that analyzes Π, identifies the rules that should be compiled and those that should be grounded, and then compiles them properly. More precisely, the *Compiler* first applies a pre-processing step that extends the normalization of ProASP to allow grounded rules to coexist with propagators, no matter if they are involved in a recursive definition. Then, the rules to be compiled follow the usual path of ProASP, i.e., they are compiled in propagators; whereas the remaining rules are compiled in a code that performs grounding, i.e., code that generates propositional clauses that are stored in the Glucose data structures. As a result, we obtain an ASP system that blends compilation and grounding.

At *Solving Stage*, instead, ProASP takes as input a set of facts F denoting a specific instance of the program Π that should be solved, and computes an answer set of $\Pi \cup F$, if any, or proves its incoherence. First of all, the input instance F is given as input to the *Hybrid Generator* module that generates the set of relevant atoms [32] for computing the answer set of $\Pi \cup F$ together with the ground instantiations of rules of Π that should be grounded. Produced ground rules are successively transformed into a set of clauses [33] that, together with generated atoms, are used to initialize the Glucose SAT solver at the core of the ProASP system. At this point, Glucose starts the CDCL for computing an answer set of $\Pi \cup F$. At each step of the CDCL algorithm, Glucose chooses a branching literal l and performs the unit propagation by (i) considering the generated clauses encoding the ground rules and (ii) using the *Propagator* module, for simulating the inferences of the remaining rules. During the CDCL, as soon as the current interpretation becomes inconsistent, the conflicting literals are analyzed according to the propagation clauses that derived them. More precisely, for all those literals that have been derived from compiled propagators, Glucose uses the *Propagator* module to reconstruct the propagation clauses, while, for all other literals, Glucose analyzes its internal clauses that caused the propagation. As soon as the current interpretation is total or the consistency cannot be restored, then the CDCL stops. At this point, if such total interpretation M has been found, then M is an answer set of $\Pi \cup F$, otherwise ProASP returns $\bot$, denoting that $\Pi \cup F$ has no answer sets.

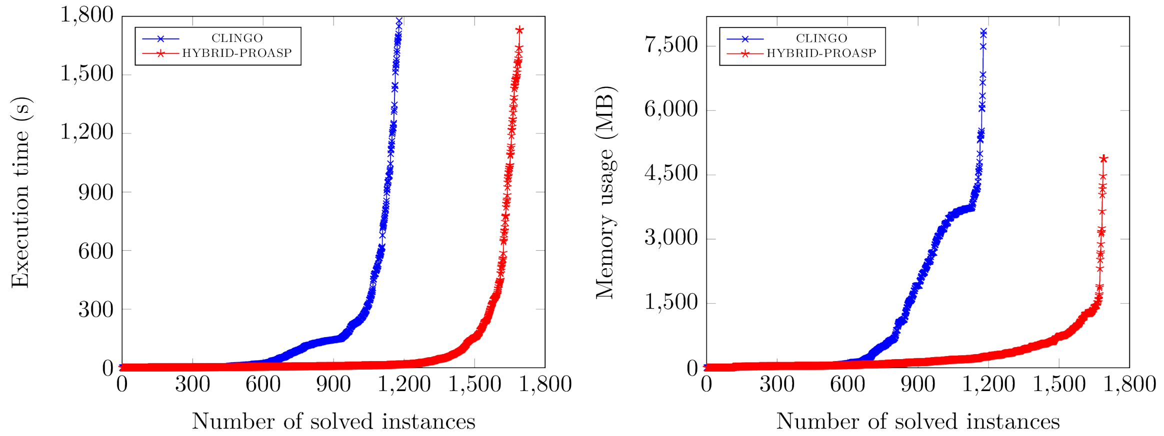

Figure 1: Comparison between *Ground&Solve* and hybrid ASP solving

3. Scalability of hybrid ASP solving

We report here about the final experiment performed in [37], where the ProASP system with hybrid solving enabled has been compared to the state-of-the-art *Ground&Solve* ASP system CLINGO on well-established benchmarks featuring 2366 instances from 14 application domains. In this evaluation, we considered eight variants of ProASP representing different ways of blending grounding and compilation by splitting programs by rule type. More precisely, we identified the following three types of rules: (CS) constraints not containing aggregates; (AG) rules containing aggregates; (RL) normal rules; and also their combinations, i.e., RL+CS, RL+AG, and CS+AG. Additionally, we also consider the two versions of ProASP in which the whole program is compiled or grounded. Finally, we label HYBRID-PROASP the ProASP version obtained by considering the best split for each benchmark. Obtained results are reported in Figure 1 where the two cactus plots report, respectively, time and memory consumption. Recall that, in a cactus plot, instances are sorted by memory (or time) usage, and a point (i, j) indicates that a solver is capable of solving the i-th instance with a memory (or time) limit of j megabytes (or seconds), respectively. As it is evident from the plots, the hybrid solving results in significant performance improvements w.r.t. the state-of-the-art by solving 513 more instances and consuming much less memory (roughly 60%). For further details, we refer reader to [37].

4. Discussion

ASP systems based on *Ground&Solve* approach are generally effective, but they can struggle with the grounding bottleneck. On the other hand, compilation-based ASP systems have shown that, in certain cases, grounding can be avoided. However, neither approach pays off in all cases.

By developing a technique that seamlessly combines grounding and compilation, we achieved notable improvements over existing compilation-based methods and current state-of-the-art implementations. This is a very promising result, expanding the applicability of ASP as an AI tool for solving complex combinatorial problems.

As far as future work is concerned, since the ProASP system does not support the entire ASP-Core 2 standard, it would be interesting to extend the benefit of the blending also to a wider class of programs such as non-tight programs and disjunctive ones. Another promising avenue is the exploration of a more fine-grained—and possibly automated—procedure for selecting which rules to ground or compile. However, this presents a significant challenge arising from the unique nature of compilation, which must be performed prior to executing any instance of the problem.

Acknowledgments

This work has been partially supported by MISE under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005, and MUR under projects: PNRR FAIR - Spoke 9 - WP 9.1 CUP H23C22000860006, Tech4You CUP H23C22000370006, and PRIN PINPOINT CUP H23C22000280006.

References

[1] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.

[2] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–386.

[3] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (2016) 53–68.

[4] T. C. Son, E. Pontelli, M. Balduccini, T. Schaub, Answer set planning: A survey, Theory Pract. Log. Program. 23 (2023) 226–298.

[5] C. Dodaro, M. Maratea, Nurse scheduling via answer set programming, in: LPNMR, volume 10377 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 301–307.

[6] M. Cardellini, C. Dodaro, G. Galatà, A. Giardini, M. Maratea, N. Nisopoli, I. Porro, Rescheduling rehabilitation sessions with answer set programming, J. Log. Comput. 33 (2023) 837–863.

[7] A. Mitra, P. Clark, O. Tafjord, C. Baral, Declarative question answering over knowledge bases containing natural language text with answer set programming, in: AAAI, AAAI Press, 2019, pp. 3003–3010.

[8] P. Schüller, Modeling variations of first-order horn abduction in answer set programming, Fundam. Informaticae 149 (2016) 159–207.

[9] Z. Yang, A. Ishay, J. Lee, Coupling large language models with logic programming for robust and general reasoning from text, in: ACL (Findings), Association for Computational Linguistics, 2023, pp. 5186–5219.

[10] T. Eiter, M. Fink, G. Greco, D. Lembo, Repair localization for query answering from inconsistent databases, ACM Trans. Database Syst. 33 (2008) 10:1–10:51.

[11] M. Arenas, L. E. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: PODS, ACM Press, 1999, pp. 68–79.

[12] M. Manna, F. Ricca, G. Terracina, Taming primary key violations to query large inconsistent data via ASP, Theory Pract. Log. Program. 15 (2015) 696–710.

[13] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: ICLP (Technical Communications), volume 52 of *OASICS*, Schloss Dagstuhl, 2016, pp. 2:1–2:15.

[14] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri, J. Zangari, The ASP system DLV2, in: LPNMR, volume 10377 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 215–221.

[15] B. Kaufmann, N. Leone, S. Perri, T. Schaub, Grounding and solving in answer set programming, AI Mag. 37 (2016) 25–32.

[16] J. Marques-Silva, I. Lynce, S. Malik, Conflict-driven clause learning SAT solvers, in: Handbook of Satisfiability, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 133–182.

[17] F. Calimeri, M. Gebser, M. Maratea, F. Ricca, Design and results of the fifth answer set programming competition, Artif. Intell. 231 (2016) 151–181.

[18] M. Ostrowski, T. Schaub, ASP modulo CSP: the clingcon system, Theory Pract. Log. Program. 12 (2012) 485–503.

[19] M. Gebser, N. Leone, M. Maratea, S. Perri, F. Ricca, T. Schaub, Evaluation techniques and systems for answer set programming: a survey, in: IJCAI, ijcai.org, 2018, pp. 5450–5456.

[20] M. Balduccini, Y. Lierler, Constraint answer set solver EZCSP and why integration schemas matter, Theory Pract. Log. Program. 17 (2017) 462–515.
[21] B. Susman, Y. Lierler, Smt-based constraint answer set solver EZSMT (system description), in: ICLP (Technical Communications), volume 52 of *OASIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 1:1–1:15.
[22] J. Bomanson, T. Janhunen, A. Weinzierl, Enhancing lazy grounding with lazy normalization in answer-set programming, in: AAAI, AAAI Press, 2019, pp. 2694–2702.
[23] C. Lefèvre, P. Nicolas, The first version of a new ASP solver : Asperix, in: LPNMR, volume 5753 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 522–527.
[24] Y. Lierler, J. Robbins, Dualgrounder: Lazy instantiation via clingo multi-shot framework, in: JELIA, volume 12678 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 435–441.
[25] A. D. Palù, A. Dovier, E. Pontelli, G. Rossi, GASP: answer set programming with lazy grounding, Fundam. Informaticae 96 (2009) 297–322.
[26] A. Weinzierl, Blending lazy-grounding and CDNL search for answer-set solving, in: LPNMR, volume 10377 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 191–204.
[27] V. Besin, M. Hecher, S. Woltran, On the structural complexity of grounding - tackling the ASP grounding bottleneck via epistemic programs and treewidth, in: ECAI, volume 372 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2023, pp. 247–254.
[28] V. Besin, M. Hecher, S. Woltran, Body-decoupled grounding via solving: A novel approach on the ASP bottleneck, in: IJCAI, ijcai.org, 2022, pp. 2546–2552.
[29] B. Cuteri, C. Dodaro, F. Ricca, P. Schüller, Partial compilation of ASP programs, Theory Pract. Log. Program. 19 (2019) 857–873. URL: https://doi.org/10.1017/S1471068419000231. doi:10.1017/S1471068419000231.
[30] B. Cuteri, C. Dodaro, F. Ricca, P. Schüller, Overcoming the grounding bottleneck due to constraints in ASP solving: Constraints become propagators, in: IJCAI, ijcai.org, 2020, pp. 1688–1694.
[31] G. Mazzotta, F. Ricca, C. Dodaro, Compilation of aggregates in ASP systems, in: AAAI, AAAI Press, 2022, pp. 5834–5841.
[32] C. Dodaro, G. Mazzotta, F. Ricca, Compilation of tight ASP programs, in: ECAI, volume 372 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2023, pp. 557–564.
[33] E. Erdem, V. Lifschitz, Tight logic programs, Theory Pract. Log. Program. 3 (2003) 499–518.
[34] K. L. Clark, Negation as failure, in: Logic and Data Bases, Advances in Data Base Theory, Plemum Press, New York, 1977, pp. 293–322.
[35] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solvers, in: IJCAI, 2009, pp. 399–404.
[36] M. Gebser, M. Maratea, F. Ricca, The seventh answer set programming competition: Design and results, Theory Pract. Log. Program. 20 (2020) 176–204.
[37] C. Dodaro, G. Mazzotta, F. Ricca, Blending Grounding and Compilation for Efficient ASP Solving, in: Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, 2024, pp. 317–328. URL: https://doi.org/10.24963/kr.2024/30. doi:10.24963/kr.2024/30.