Proceedings of the Sixteenth International Conference on
Principles of Knowledge Representation and Reasoning (KR 2018)

Towards Lazy Grounding with Lazy Normalization in Answer-Set Programming — Extended Abstract

Jori Bomanson, Tomi Janhunen, Antonius Weinzierl
Department of Computer Science, Aalto University, Finland

Abstract

Answer-Set Programming (ASP) is an expressive rule-based knowledge-representation formalism supported by efficient solver technology. Traditional evaluation of answer-set programs takes place in two phases: grounding and solving. Grounding incurs an up-to exponential increase in space, termed the grounding bottleneck of ASP, which is often encountered in practice. Lazy grounding avoids this bottleneck but is restricted to normal rules, significantly limiting the expressive power of this approach. We propose a framework to handle aggregates by normalizing them on demand during the lazy grounding process; we call this approach lazy normalization. It is feasible for different types of aggregates and can bring about up-to exponential gains in space and time.

1 Introduction

Answer-Set Programming (ASP) is an expressive rule-based knowledge-representation formalism whose success is much due to efficient solver technology (Gebser, Kaufmann, and Schaub 2012; Alviano et al. 2013; Leone et al. 2002). State-of-the-art ASP systems adhere to the ground-and-solve approach, where a first-order input program is turned into a corresponding *ground* (variable-free) program for which answer sets are then computed. But, in the worst case, this ground program may be exponentially larger than the original non-ground input program. Even a polynomial-size increase may already be prohibitive in practice. This drawback impairs the scalability of ASP for practical applications (cf. (Falkner et al. 2016)) and is known as the grounding bottleneck of ASP. To circumvent such blow-ups, *lazy-grounding* ASP solvers have been developed; cf. (Palù et al. 2009; Lefèvre et al. 2017; Dao-Tran et al. 2012; Weinzierl 2017). The main idea of lazy grounding is to interleave the grounding and solving phases and to generate only ground rules necessary in each position of the search space.

However, the existing lazy-grounding ASP systems only accept *normal rules* as input and they do not support a broad range of syntactic primitives as defined by, e.g., the ASP *core language* (Calimeri et al. 2012). Most notably missing are *aggregates*, which occur in many ASP programs, because they are highly expressive and enable a programmer to state complex conditions in a very concise manner. The importance of aggregates is witnessed by a rich body of research, see e.g., (Greco 1999; Simons, Niemelä, and Soininen 2002; Liu and Truszczynski 2006; Ferraris 2011; Faber, Pfeifer, and Leone 2011; Gelfond and Zhang 2014; Alviano, Faber, and Gebser 2015). Realizing aggregates is feasible as native extensions of solvers or by unfolding (monotone) ground aggregates as normal rules. The latter is known as *normalization* (Janhunen and Niemelä 2011; Bomanson and Janhunen 2013). Normalization is an appealing technique since existing solving techniques can be deployed directly, e.g., conflict-driven learning across normalized aggregates is achieved for free. Furthermore, it easily allows to revise encodings of aggregates in systematic fashion. The integration of aggregates into lazy-grounding ASP systems, however, has not been attempted so far, although there is some work on first-order rewriting of aggregates for ground-and-solve ASP systems (Polleres et al. 2013).

2 Lazy Normalization

To illustrate the idea of lazy normalization, let us consider the following first-order rule with a counting aggregate:

$$ok \leftarrow 1 \leq \#count\{1 : p(X), d(X)\}.$$

Assuming that the only facts present are $d(a)$, $d(b)$, and $d(c)$, the traditional grounding of this rule yields the ground rule: $ok \leftarrow 1 \leq \#count\{1 : p(a);\ 1 : p(b);\ 1 : p(c)\}$. If further normalized, we obtain the normal rules $ok \leftarrow p(a)$, $ok \leftarrow p(b)$, and $ok \leftarrow p(c)$. In this work, however, normalization is applied at the first-order level and the result expresses the aggregated rule with normal first-order rules. Due to the special bound 1 of the above aggregate, the original rule can be normalized as a first-order rule $ok \leftarrow p(X), d(X)$. The lazy grounding of this rule then amounts to the *lazy normalization* of the original rule, because the respective ground normal rules are produced lazily and only when needed. It is worth emphasizing that for bounds greater than 1, the normalization of rules involving counting aggregates at the first-order level becomes far more involved.

Indeed, lazy grounding poses some unexpected challenges to normalization, because matters like enumerating all relevant ground instances of a variable, which are trivial in the ground-and-solve approach, suddenly become challenging: for every variable X it is unclear what the ground instances are, how many of them will appear, and in what order. On the one hand, counting inherently requires that the counted ground atoms are totally ordered; on the other hand, for an aggregate counting the cardinality of $p(X)$ it is not known whether the ground instances $p(a)$ and $p(c)$ will be grounded lazily at some point, nor is it then known if $p(c)$ is the second ground instance since some later grounded $p(b)$ might come between $p(a)$ and $p(c)$ in a natural order. Thus it is not obvious how normalization can be efficiently applied in a lazy-grounding setting. Another challenge is that the normalization of an aggregate must not require any portion of the program to be fully grounded, since due to predicate dependency this can easily require large portions of the program to be fully grounded and hence degenerate into grounding the input program completely.

Our investigation revealed that ground instance enumeration is a key principle in enabling efficient non-ground normalization in a lazy-grounding setting. Based on this we developed a normalization framework working in the context of lazy grounding, which we coin lazy normalization since it allows for the result of normalization to be instantiated lazily. Lazy normalization of aggregates with lower bounds is possible, and an optimized lazy normalization for certain aggregates, inheriting attractive properties from ground-and-solve normalizations, was achieved. The normalizations have been implemented within the Alpha ASP system, which performed well in several benchmarks, allowing significant savings in space and time. Furthermore, this work enables an unprecedented lazy-grounding ASP evaluation beyond normal rules and a significant step towards the full expressive power of ASP in lazy-grounding ASP systems and related hybrid approaches, such as (Eiter, Kaminski, and Weinzierl 2017).

Acknowledgments

This work has been supported by the Academy of Finland project 251170.

References

Alviano, M.; Dodaro, C.; Faber, W.; Leone, N.; and Ricca, F. 2013. WASP: A native ASP solver based on constraint learning. In *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013. Proceedings*, 54–66.

Alviano, M.; Faber, W.; and Gebser, M. 2015. Rewriting recursive aggregates in answer set programming: back to monotonicity. *TPLP* 15(4-5):559–573.

Bomanson, J., and Janhunen, T. 2013. Normalizing cardinality rules using merging and sorting constructions. In *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013. Proceedings*, 187–199.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Ricca, F.; and Schaub, T. 2012. ASP-CORE-2 input language format.

Dao-Tran, M.; Eiter, T.; Fink, M.; Weidinger, G.; and Weinzierl, A. 2012. Omiga : An open minded grounding on-the-fly answer set solver. In *Logics in Artificial Intelligence - 13th European Conference, JELIA 2012. Proceedings*, 480–483.

Eiter, T.; Kaminski, T.; and Weinzierl, A. 2017. Lazy-grounding for answer set programs with external source access. In *Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017. Proceedings*, 1015–1022.

Faber, W.; Pfeifer, G.; and Leone, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.* 175(1):278–298.

Falkner, A. A.; Friedrich, G.; Haselböck, A.; Schenner, G.; and Schreiner, H. 2016. Twenty-five years of successful application of constraint technologies at Siemens. *AI Magazine* 37(4):67–80.

Ferraris, P. 2011. Logic programs with propositional connectives and aggregates. *ACM Trans. Comput. Log.* 12(4):25:1–25:40.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.* 187:52–89.

Gelfond, M., and Zhang, Y. 2014. Vicious circle principle and logic programs with aggregates. *TPLP* 14(4-5):587–601.

Greco, S. 1999. Dynamic programming in datalog with aggregates. *IEEE Trans. Knowl. Data Eng.* 11(2):265–283.

Janhunen, T., and Niemelä, I. 2011. Compact translations of non-disjunctive answer set programs to propositional clauses. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *LNCS*, 111–130. Springer.

Lefèvre, C.; Beatrix, C.; Stephan, I.; and Garcia, L. 2017. Asperix, a first-order forward chaining approach for answer set computing. *TPLP* 1 45.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2002. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic* 7:499–562.

Liu, L., and Truszczynski, M. 2006. Properties and applications of programs with monotone and convex constraints. *J. Artif. Intell. Res.* 27:299–334.

Palù, A. D.; Dovier, A.; Pontelli, E.; and Rossi, G. 2009. Gasp: Answer set programming with lazy grounding. *Fundam. Inform.* 96(3):297–322.

Polleres, A.; Frühstück, M.; Schenner, G.; and Friedrich, G. 2013. Debugging non-ground ASP programs with choice rules, cardinality and weight constraints. In *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013. Proceedings*, 452–464.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artif. Intell.* 138(1-2):181–234.

Weinzierl, A. 2017. Blending lazy-grounding and CDNL search for answer-set solving. In *Logic Programming and Nonmonotonic Reasoning, 14th International Conference, LPNMR 2017. Proceedings*, 191–204.