The Alpha Solver for Lazy-Grounding Answer-Set Programming

Antonius Weinzierl[1], Bart Bogaerts[2], Jori Bomanson[3], Thomas Eiter[1], Gerhard Friedrich[4], Tomi Janhunen[3,5], Tobias Kaminski[1], Michael Langowski[1], Lorenz Leutgeb[1], Gottfried Schenner[6], and Richard Taupe[4,6]

[1] Insitute of Logic and Computation, Vienna University of Technology, Austria
[2] Department of Computer Science, Vrije Universiteit Brussel (VUB), Belgium
[3] Department of Computer Science, Aalto University, Finland
[4] Department of Applied Informatics, Alpen-Adria-Universität Klagenfurt, Austria
[5] Information Technology and Communication Sciences, Tampere University, Finland
[6] Corporate Technology, Siemens AG Österreich

`weinzierl@kr.tuwien.ac.at`

Abstract. The grounding bottleneck is a longstanding issue of Answer-Set Programming (ASP), a well-known Logic Programming formalism widely used for declarative problem solving. Lazy grounding as realized by the recent Alpha system avoids this grounding bottleneck but faces new challenges that are genuine to lazy grounding. This article first gives an overview of lazy-grounding ASP solving by Alpha and provides information on how to obtain and use the system. It then presents research issues raised by lazy-grounding and overviews those which have been addressed already.

1 Introduction

Answer-Set Programming (ASP) [4, 19, 28, 34, 35] is a well-known Logic Programming formalism, widely used for declarative problem solving in diverse areas, ranging from planning, optimization, and commonsense reasoning to explanation finding. The success of ASP is rooted in efficient solvers, due to which ASP is applied fruitfully in a broad range of applications, from NASA's Space Shuttle [33, 48], to planning in the automotive industry [32], configuration [24], etc [9, 52]. Despite this success, ASP solving has always been hampered by the grounding bottleneck, which is an inherent drawback of the traditional ground-and-solve evaluation of ASP that is employed by many of the most prominent ASP solvers till today. First, a non-ground (i.e., first-order) ASP program is fully grounded upfront to obtain a variable-free (i.e., propositional) program. In a second step, the answer sets of the grounded program are computed using efficient, propositional solving techniques adapted or inspired from SAT solving [1, 29, 41]. As the grounded program may be exponentially larger than the original ASP program, the ensuing memory blow-up renders the ground-and-solve approach categorically inapplicable for entire classes of programs.

Lazy grounding avoids upfront grounding and the grounding bottleneck by interleaving the solving and the grounding phases. While ground-and-solve systems have to ground all rules that may fire in any state the solver possibly reaches, in lazy grounding only those rules are grounded that may fire in the current state of the solver. If the solver recognizes that some part of the search space contains no answer sets, lazy-grounding can omit grounding the rules that only fire in that part. There are several systems for lazy-grounding ASP solving available, namely GASP [49], ASPeRiX [38–40], Omiga [13, 57], and Alpha [58], which are all based on the notion of a computation sequence [43]. Unfortunately, lazy-grounding cannot be simply put on top of the traditional state-of-the-art techniques for efficient ASP solving, since one of their core-assumptions, that all relevant ground rules are known, does not apply to lazy-grounding. The latter thus opens many issues and novel research questions, some of which have been closed by transferring techniques from the ground-and-solve approach to the lazy-grounding setting. In particular, Alpha is the first system to combine lazy-grounding with efficient search techniques from traditional ground-and-solve systems, such as conflict-driven learning and watched-literals propagation. Although its search performance does not yet match the best possible with ground-and-solve systems, Alpha is a big improvement compared to other lazy-grounding systems and offers the following features:

- Computation of answer sets without the need for upfront grounding, which avoids the grounding bottleneck of ASP and thus allows to solve ASP programs where traditional solvers run out of memory;
- A combination of lazy-grounding with conflict-driven learning and other techniques for efficient (propositional) ASP solving;
- First-order normalizations to evaluate aggregates with Alpha, while its core works with normal rules;
- Non-ground justifications for atoms, to avoid a genuine problem of lazy grounding where the search may get stuck if not all ground rules that potentially can derive an atom are known;
- Heuristics specifically tailored for lazy grounding;
- An integration with the HEX framework;
- A free and open-source implementation in Java.

Overall, Alpha already strikes a good balance between efficient grounding and solving, while there are several open problems with potential for future research. The active development of Alpha is continuing and we would welcome new users, feedback on the software, and collaboration on open issues.

In this article we review the basics underlying Alpha, present research advances to improve lazy-grounding ASP solving, and discuss open research questions. The remainder of this article is thus structured as follows. Section 2 shows where to get and how to run Alpha, while Section 3 gives an overview of the system and its components. Section 4 introduces research issues related to lazy grounding that already have been addressed, whereas Section 5 presents ongoing and open issues. Section 6 considers related work and, finally, Section 7 concludes this article.

2 Obtaining and Running Alpha

The Alpha system and its source code are publicly available on GitHub.[1] Alpha is actively developed and new binary releases are made available in irregular intervals. All releases can be downloaded in compiled form as Java Archive (JAR) files. These files can be run on all major operating systems via the Java runtime (version 8 or higher) by executing `java -jar alpha-bundled.jar`. Invoking Alpha without any arguments like this will print an overview of arguments and exit. Building the newest version from source is possible using the automated build system Gradle[2] and detailed instructions are available on GitHub.[1]

Example 1. Let `encoding.lp` be a problem encoding in ASP and `inst43.lp` contain an instance, i.e., a number of facts. Then Alpha can be instructed to search the first 5 answer sets as follows:

```
java -jar alpha-bundled.jar -n 5 -i encoding.lp -i inst43.lp
```

One can also specify parts of the input program directly from the command line, which is useful for setting parameters like domain size or bounds of an ASP program.

Example 2. Assume the following ASP program to compute Fibonacci numbers up to a given number is given in a file `fib.lp`.

```
fib(0,0). fib(1,1).
fib(N,F) :- fib(N1,F1), fib(N2,F2), N1=N-1, N2=N-2, F = F1+F2,
            N=0..U, upto(U).
```

The command line to start Alpha and compute Fibonacci numbers up to 22 is:

```
java -jar alpha-bundled.jar -i fib.lp -str "upto(22)."
```

The input language currently accepted by Alpha is a subset of the ASP-Core-2 input language [11] including function symbols and interval terms. Since the core of Alpha works with normal rules, we employ normalizations to transform programs using richer syntax into normal rules. At the time of writing, a growing list of supported constructs includes: choice rules without bounds, sum and count aggregates with lower bounds, and arithmetic terms in comparison relations. Full support for ASP-Core-2 is planned and ongoing. As usual in ASP, rules must be safe, i.e., every variable of a rule must also occur in its positive body.

Comments and suggestions, as well as contributions via GitHub are very much encouraged and welcomed.

3 Overview of the Alpha System

Alpha follows the notion of ASP computation for lazy-grounding and combines it with the traditional techniques for efficient ASP evaluation, where a dedicated

[1] https://github.com/alpha-asp/alpha
[2] https://gradle.org/

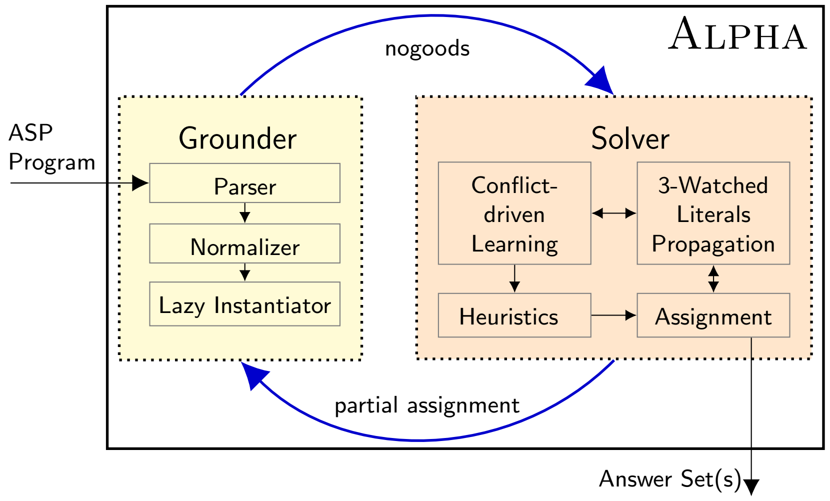

Fig. 1. The lazy-grounding ASP system Alpha.

solving component employs SAT-inspired techniques to find answer sets of a set of ground nogoods and another component produces ground nogoods from the input ASP program. Different from traditional ASP solving, however, these components interact in a cycle and the algorithms in the solving component are aware of the fact that they see only a part of the full grounding of the input program.

Figure 1 gives an overview of the Alpha system and its components. There are two major components: the Grounder and the Solver. The Grounder first parses and normalizes the input program and then grounds it lazily. The Solver uses techniques for efficient ASP solving, conflict-driven learning, watched-literals propagation, and heuristics as used by state-of-the-art ASP systems, but customized for lazy grounding [58]. At startup, the Grounder derives ground rules that are applicable under the given input facts, turns them into nogoods (i.e., constraints that must be satisfied) and hands the latter on to the Solver. The Solver in turn employs a CDCL-style algorithm to derive (or extend) a partial assignment, which is then given to the Grounder to obtain further applicable ground rules, whose nogood representation is again given to the Solver.

4 Research Topics Addressed

Since 2016, when work on Alpha started, we have encountered a number of research issues and we have achieved significant progress on multiple fronts that we summarize below.

Efficient Propagation. A key issue for ASP solving is propagation in CDCL-based search; as this is frequently executed, much effort has been spent to come up with an efficient solution. The ALPHA system uses three truth values (*must-be-true* in addition to *true* and *false*) and therefore implements a generalization of the well-known two-watched literals (2WL) scheme commonly employed in SAT and ASP solving, which we call the *three-watched literals scheme* [42]. The latter introduces notions of *weakly unit nogoods* (propagating *false* or *must-be-true*) and *strongly unit nogoods* (which must have a designated head literal for propagating any truth value). Here, propagation from *must-be-true* to *true* always occurs by the head of a nogood, which requires a third watched literal. The solver maintains a *watch structure* for every atom, which points to the nogoods where the respective atom is being watched. This watch structure then is used for propagation in computation towards a least fixpoint.

Aggregate Rewriting. Recently, the *ASP-Core-2* support in ALPHA broadened to include threshold conditions on the minimum, count, and sum of terms via the powerful and highly expressive language construct of *aggregates*. In appreciation of the integral role of aggregates in practice and research [53, 2, 23], ALPHA delivered monotone count and sum aggregate support to the lazy-grounding ASP realm [7, 8]. The approach instantiates a novel framework of *lazy normalization* in translating aggregates to efficient first-order normal-rule encodings. The approach overcomes key challenges posed by the lazy grounding of aggregates: it hides the complexity of aggregates from the core of the solver, which already faces the tremendous challenge and complexity of both grounding and solving. Moreover, efficient lazy grounding precludes upfront counting and ordering of predicate instances. Yet, usual normalization depends crucially on both abilities [6]. This dilemma motivated the innovation of a novel language primitive that exposes a lazily determined order of atoms and enables asymptotically more concise normalizations than previously known on the first-order level [50].

Justifications. Whenever ALPHA arrives in a situation where a true atom, say p, is unjustified (in the terminology from above, this means that p is must-be-true but not true) and no further choices or propagations are possible, the basic ALPHA algorithm resorts to chronological backtracking. This turned out to be problematic in several applications; to avoid this, we devised a method that analyses *why* p is not justified and subsequently learns a new clause that intuitively states that whenever p is true, the discovered reason for p being unjustified should be invalidated [5]. Formally this analysis builds on the recently introduced theory of justifications [14]. In practice, our algorithm — inspired by partial evaluation techniques [37] — performs a top-down analysis of the logic program starting from p to find all literals that (directly or indirectly) blocked p from being justified, either by falsifying a rule body of a rule known to the solver that could contribute to a justification of p or by blocking a rule from being grounded. Preliminary experiments are promising and show that up to exponential speed-ups can be gained and that on previously existing benchmarks, the justification analysis eliminates a need for addition of redundant constraints.

Search Heuristics. Search heuristics steer solvers through the search space and may shorten the search by a large amount. Alpha takes ideas from domain-independent search heuristics like *VSIDS* [47] and *BerkMin* [36], which were originally developed for SAT but are also successfully employed by ASP solvers (such as clasp [31] and wasp [1]). These heuristics take into account how often and how recently a propositional variable contributed to a conflict.

A direct application of *BerkMin* or *VSIDS* to lazy-grounding systems like Alpha is problematic because their solving algorithms are quite different from those in ground-and-solve systems. A major difference is that not all ground rules, and consequently not all ground atoms, are known to a lazy-grounding solver at any time. A further difference is that while a traditional ASP solver can choose any atom to guess on, Alpha only guesses on atoms representing bodies of rules that almost fire in the sense that the positive body is true already and the negative body is not falsified. Therefore, for Alpha a set of novel domain-independent heuristics has been developed [56]. First benchmarks are promising but also indicate the need for further study.

Integration with HEX. The hex-*formalism* extends ASP by allowing a bidirectional exchange between programs and external computation sources, which are interfaced via so-called *external atoms* [18]. For instance, an external atom $\&onlineWeather[loc](X)$ may query an online weather service for the weather report for all locations in the extension of the *loc* predicate. In practice, external atoms are realized by C++ or Python plug-ins in an API-style fashion, and the formalism has been applied to a wide range of areas ranging from Semantic Web applications to route planning [20]. A longstanding issue regarding hex-evaluation has been the grounding of programs containing nonmonotonic external atoms that introduce new constants by so-called *value invention* as such atoms need to be evaluated under exponentially many inputs during grounding. This makes the grounding for hex even more challenging than for ordinary ASP.

Alpha enabled the integration of grounding and solving of hex-programs such that new output constants of external atoms can be generated on-the-fly during solving [21]. For this, Alpha has been integrated as a backend-solver into the dlvhex system, which resulted in a novel algorithm for evaluating hex-programs based on lazy grounding. Using Alpha solved the issue with respect to grounding nonmonotonic external atoms and for grounding-intense programs, a clear advantage of the Alpha-based algorithm could be shown in practice.

5 Ongoing and Future Work

The development of Alpha is ongoing and researchers from multiple universities and the industry are collaborating to further improve the state of lazy-grounding ASP solving. We present some of the currently ongoing work related to Alpha below.

Grounding Strategies. Alpha was equipped with state-of-the art heuristics successfully employed by other ASP solvers, namely MOMs [51] for initialization of heuristic scores and VSIDS [47] for their dynamic modification. Both are implemented in a similar fashion as in CLASP [31]. However, the performance improvement by those heuristics was much smaller than expected, because lazy grounding provides a too narrow view of the search space for such heuristics to perform adequately. This is a novel challenge for ASP solving, which traditional ground-and-solve ASP solvers did not have to face.

So far ALPHA runs a very restrictive grounding strategy in order to save maximum space, which results in non-optimal search performance as state-of-the-art search heuristics are left mostly blind because they only consider the grounded information. Thus we investigated more permissive lazy-grounding strategies that ground more than what is absolutely neccessary. They produce ground rules earlier, which informs search procedures better about the problem at hand. For more details see the upcoming paper [55].

Domain-Specific Heuristics. A major advance in solving industrial configuration problems with ASP can be achieved with domain-specific heuristics (cf. [16, 26, 30]), which allow to use heuristics that are designed specifically for one given application domain. Given, e.g., a bin-packing problem, a domain-specific heuristic for the solver could be to pick the biggest item not yet placed and put it in a fitting bin with the least space remaining (i.e., a best-fit decreasing heuristic).

In [54], a novel semantics for heuristic directives in ASP is presented that allows declarative specification of domain-specific heuristics where default negation inside heuristic conditions holds for *false* and *currently unassigned* atoms. This allows for a natural and declarative formalisation of many domain-specific heuristics, e.g., in a bin-packing encoding to refer to items not yet placed or the total weight of items already placed in the partial assignment. The implementation of such heuristics in ALPHA is currently ongoing.

Partial Evaluation. Traditional ground-and-solve ASP systems evaluate the definite part of the given input program already in the grounding phase in order to reduce the instance of the subsequent solving phase. In ALPHA there is ongoing work to realize partial evaluation in a similar way, based on the part of the input that can be stratified [3] and which does not depend on any choices to be made by the solver. For this, ALPHA analyses the dependency graph of the input program on a predicate level, identifies strongly connected components (SCCs), and then evaluates the acyclic part that does not depend on any cycles. This approach is not as general as, e.g. the one in DLV [22] which considers dependency on ground rules; however, it enables a subsequent search strategy based on SCCs similar to ASPERIX.

Future Work. A medium-term goal for ALPHA is to support the full ASP-Core-2 input language, which includes weak constraints, optimization, and disjunction. A further such goal is to transfer all major techniques for efficient answer-set solving, like learned clause deletion, rapid restarts, etc., to the lazy-grounding setting. For many of them, lazy-grounding raises novel research questions and issues that do not exist if the full grounding is available. They pose completely new challenges and open avenues for novel solving techniques that make use of the first-order representation of the program. We will continue to investigate these avenues and invite the community to join our efforts.

6 Related Work

Alpha is the latest in a line of lazy-grounding ASP solvers being based on the notion of a computation sequence [43]. GASP [49] and ASPeRiX [39, 40, 38] were the first lazy-grounding ASP solvers that implemented this notion. Given a (partial) interpretation, finding a ground instance of a non-ground rule, i.e., grounding the rule lazily, is a task that is well-known to be NP-complete. For this reason, lazy grounding inside GASP was encoded as a constraint problem, while ASPeRiX used a semi-naive grounding approach. The later Omiga solver [13, 57] used a RETE network [25] to speed-up this task but otherwise followed the idea of a computation sequence quite closely. As noted in [38], however, semi-naive grounding seems to be sufficiently fast for lazy-grounding ASP solving.

Computation sequences have also been implemented on top of special hardware, in particular graphics cards to make use of their massive parallelization [17]. This approach, however, uses no lazy grounding but the traditional full upfront grounding.

Another approach to avoid grounding is based on a top-down query-driven method to evaluate normal logic programs in a way similar to Prolog and with negation as failure under the stable model semantics [44]. Galliwasp [45] is the propositional stepping stone for the s(ASP) system [46], which implements this query-driven ASP evaluation. Similar to Alpha, it does not require a finite grounding, but the s(ASP) prototype, in contrast, is not designed to be competitive in terms of speed and no benchmark results for search performance beyond trivial problem instances have been reported.

Goal-driven lazy grounding is also realized in the Lazy-MX [12] system, which achieves efficient solving performance for its language of $FO(ID)$ that essentially corresponds to the generate-define-test fragment of ASP [15].

Another way to avoid the grounding bottleneck of ASP is to outsource difficult-to-ground parts of a problem specification. Recent applications with Clingo [27] represent parts of a problem in an extra theory and employ the ASP modulo theories approach to combine the outsourced theory reasoning with the parts specified in ASP. This is akin to SAT modulo theories (SMT). The obvious drawback of such an approach, however, is that the problem now has to be specified in two distinct formalisms and for solving, these formalisms must be combined again in a suitable manner.

7 Conclusion

We have presented the Alpha system for answer-set solving, which avoids the grounding bottleneck using a lazy-grounding approach. While on programs whose full upfront grounding easily fits into memory the solving performance of Alpha is not yet on par with the best performance offered by traditional ground-and-solve ASP systems, Alpha offers competitive solving performance on programs where grounding is demanding, because it implements the most important techniques for efficient ASP solving already. By that, Alpha makes ASP solving feasible for entirely new classes of programs and new application domains from industry.

The system is ready to be used and ongoing development aims at supporting the full ASP-Core-2 input language as well as improved solving performance. Alpha and its source code are freely available on GitHub.[3] We are happy to receive bug reports and would very much welcome potential future contributors and collaborators.

Acknowledgements

Research on Alpha has been supported by the Austrian Science Fund (FWF) projects P27730 and W1255, the Academy of Finland, project 251170, and the research project *DynaCon (FFG-PNr.: 861263)*, which is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between 2017 and 2020 (for more information see https://iktderzukunft.at/en/).

References

1. Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In Cabalar and Son [10], pages 54–66.
2. Mario Alviano, Wolfgang Faber, and Martin Gebser. Rewriting recursive aggregates in answer set programming: back to monotonicity. *TPLP*, 15(4-5):559–573, 2015.
3. Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Chapter 2 - towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89 – 148. Morgan Kaufmann, 1988.
4. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
5. Bart Bogaerts and Antonius Weinzierl. Exploiting justifications for lazy grounding of answer set programs. In *IJCAI*, pages 1737–1745, 2018.
6. Jori Bomanson and Tomi Janhunen. Normalizing cardinality rules using merging and sorting constructions. In Cabalar and Son [10], pages 187–199.
7. Jori Bomanson, Tomi Janhunen, and Antonius Weinzierl. Towards lazy grounding with lazy normalization in answer-set programming - extended abstract. In *KR*, pages 625–626. AAAI Press, 2018.
8. Jori Bomanson, Tomi Janhunen, and Antonius Weinzierl. Enhancing lazy grounding with lazy normalization in answer-set programming. In *AAAI*. AAAI Press, 2019. To appear.

[3] https://github.com/alpha-asp/alpha

9. Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming: An introduction to the special issue. *AI Magazine*, 37(3):5–6, 2016.
10. Pedro Cabalar and Tran Cao Son, editors. *LPNMR*, volume 8148 of *LNCS*, 2013.
11. Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-core-2 input language format, March 2012.
12. Broes De Cat, Marc Denecker, Maurice Bruynooghe, and Peter J. Stuckey. Lazy model expansion: Interleaving grounding with search. *J. Artif. Intell. Res.*, 52:235–286, 2015.
13. Minh Dao-Tran, Thomas Eiter, Michael Fink, Gerald Weidinger, and Antonius Weinzierl. Omiga: An open minded grounding on-the-fly answer set solver. In *JELIA*, volume 7519 of *LNCS*, pages 480–483. Springer, 2012.
14. Marc Denecker, Gerhard Brewka, and Hannes Strass. A formal theory of justifications. In *LPNMR*, pages 250–264, 2015.
15. Marc Denecker, Yuliya Lierler, Miroslaw Truszczynski, and Joost Vennekens. The informal semantics of answer set programming: A tarskian perspective. *CoRR*, abs/1901.09125, 2019.
16. Carmine Dodaro, Philip Gasteiger, Nicola Leone, Benjamin Musitsch, Francesco Ricca, and Konstantin Schekotihin. Combining Answer Set Programming and domain heuristics for solving hard industrial problems (Application Paper). *TPLP*, 16(5-6):653–669, 2016.
17. Agostino Dovier, Andrea Formisano, Enrico Pontelli, and Flavio Vella. A GPU implementation of the ASP computation. In *PADL*, volume 9585 of *LNCS*, pages 30–47. Springer, 2016.
18. 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(4):418–464, 2016.
19. Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web*, volume 5689 of *LNCS*, pages 40–110. Springer, 2009.
20. Thomas Eiter, Tobias Kaminski, Christoph Redl, Peter Schüller, and Antonius Weinzierl. Answer set programming with external source access. In *Reasoning Web*, volume 10370 of *LNCS*, pages 204–275. Springer, 2017.
21. Thomas Eiter, Tobias Kaminski, and Antonius Weinzierl. Lazy-grounding for answer set programs with external source access. In *IJCAI*, pages 1015–1022. ijcai.org, 2017.
22. Wolfgang Faber, Nicola Leone, and Simona Perri. The intelligent grounder of DLV. In *Correct Reasoning*, volume 7265 of *LNCS*, pages 247–264. Springer, 2012.
23. Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.
24. Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen. *Knowledge-based configuration: From research to business cases*. Morgan Kaufmann, 2014.
25. Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982.
26. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, and Sven Thiele. *Potassco User Guide version 2.1.0*, 2017.
27. 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 - Leibniz-Zentrum fuer Informatik, 2016.

28. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.
29. Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp*: A conflict-driven answer set solver. In *LPNMR*, pages 260–265, 2007.
30. Martin Gebser, Benjamin Kaufmann, Ramón Otero, Javier Romero, Torsten Schaub, and Philipp Wanko. Domain-specific heuristics in answer set programming. In *AAAI*, pages 350–356. AAAI Press, 2013.
31. Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012.
32. Martin Gebser, Philipp Obermeier, Torsten Schaub, Michel Ratsch-Heitmann, and Mario Runge. Routing driverless transport vehicles in car assembly with answer set programming. *TPLP*, 18(3-4):520–534, 2018.
33. Michael Gelfond. The usa-advisor: A case study in answer set programming. In Sergio Flesca, Sergio Greco, Giovambattista Ianni, and Nicola Leone, editors, *Logics in Artificial Intelligence*, pages 566–567, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
34. Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.* Cambridge University Press, New York, NY, USA, 2014.
35. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.
36. Evguenii I. Goldberg and Yakov Novikov. Berkmin: A fast and robust sat-solver. In *DATE*, pages 142–149. IEEE Computer Society, 2002.
37. Henryk Jan Komorowski. An introduction to partial deduction. In *META*, pages 49–69, 1992.
38. Claire Lefèvre, Christopher Beatrix, Igor Stephan, and Laurent Garcia. Asperix, a first-order forward chaining approach for answer set computing. *TPLP*, page 1–45, Jan 2017.
39. Claire Lefèvre and Pascal Nicolas. A first order forward chaining approach for answer set computing. In *LPNMR*, volume 5753 of *LNCS*, pages 196–208, 2009.
40. Claire Lefèvre and Pascal Nicolas. The first version of a new ASP solver: Asperix. In *LPNMR*, volume 5753 of *LNCS*, pages 522–527, 2009.
41. Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7:499–562, 2002.
42. Lorenz Leutgeb and Antonius Weinzierl. Techniques for efficient lazy-grounding ASP solving. In *DECLARE*, volume 10997 of *LNCS*, pages 132–148. Springer, 2017.
43. Lengning Liu, Enrico Pontelli, Tran Cao Son, and Miroslaw Truszczynski. Logic programs with abstract constraint atoms: The role of computations. In *ICLP*, pages 286–301, 2007.
44. Kyle Marple, Ajay Bansal, Richard Min, and Gopal Gupta. Goal-directed execution of answer set programs. In Danny De Schreye, Gerda Janssens, and Andy King, editors, *PPDP*, pages 35–44. ACM, 2012.
45. Kyle Marple and Gopal Gupta. Galliwasp: A goal-directed answer set solver. In *Proceedings of LOPSTR, Revised Selected Papers*, pages 122–136, 2012.
46. Kyle Marple, Elmer Salazar, and Gopal Gupta. Computing stable models of normal logic programs without grounding. *CoRR*, abs/1709.00501, 2017.

47. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535. ACM, 2001.
48. Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An a-prolog decision support system for the space shuttle. In *International symposium on practical aspects of declarative languages*, pages 169–183. Springer, 2001.
49. Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. Gasp: Answer set programming with lazy grounding. *Fundam. Inform.*, 96(3):297–322, 2009.
50. Axel Polleres, Melanie Frühstück, Gottfried Schenner, and Gerhard Friedrich. Debugging non-ground ASP programs with choice rules, cardinality and weight constraints. In Cabalar and Son [10], pages 452–464.
51. Daniele Pretolani. Efficiency, and stability of hypergraph SAT algorithms. In *Cliques, Coloring, and Satisfiability*, volume 26, pages 479–498. DIMACS/AMS, 1993.
52. Torsten Schaub and Stefan Woltran. Special issue on answer set programming. *KI*, 32(2-3):101–103, 2018.
53. Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.
54. Richard Taupe, Konstantin Schekotihin, Peter Schüller, Antonius Weinzierl, and Gerhard Friedrich. Towards exploiting partial knowledge in declarative domain-specific heuristics for ASP. In *Workshop on Trends and Applications of Answer Set Programming*, 2018.
55. Richard Taupe, Antonius Weinzierl, and Gerhard Friedrich. Degrees of laziness in grounding: Effects of lazy-grounding strategies on ASP solving. In *LPNMR*, 2019. To appear.
56. Richard Taupe, Antonius Weinzierl, and Gottfried Schenner. Introducing heuristics for lazy-grounding ASP solving. In *1st International Workshop on Practical Aspects of Answer Set Programming*, 2017.
57. Antonius Weinzierl. Learning non-ground rules for answer-set solving. In *Grounding and Transformation for Theories with Variables*, pages 25–37, 2013.
58. Antonius Weinzierl. Blending lazy-grounding and CDNL search for answer-set solving. In *LPNMR*, volume 10377 of *LNCS*, pages 191–204. Springer, 2017.