Speeding up Lazy-Grounding Answer Set Solving

Richard Taupe
Alpen-Adria-Universität, Klagenfurt, Austria
Siemens AG Österreich, Vienna, Austria
rtaupe@edu.aau.at
https://orcid.org/0000-0001-7639-1616

Abstract

The grounding bottleneck is an important open issue in Answer Set Programming. Lazy grounding addresses it by interleaving grounding and search. The performance of current lazy-grounding solvers is not yet comparable to that of ground-and-solve systems, however. The aim of this thesis is to extend prior work on lazy grounding by novel heuristics and other techniques like non-ground conflict learning in order to speed up solving. Parts of expected results will be beneficial for ground-and-solve systems as well.

2012 ACM Subject Classification Computing methodologies → Logic programming and answer set programming

Keywords and phrases answer set programming, lazy grounding, heuristics

Digital Object Identifier 10.4230/OASIcs.ICLP.2018.20

Funding This work was partially conducted within the scope of 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 November 2017 and April 2020.
More information: `https://iktderzukunft.at/en/`

Acknowledgements Thanks to my advisor Prof. Gerhard Friedrich, to my close collaborators Andreas Falkner, Antonius Weinzierl, Gottfried Schenner, Konstantin Schekotihin, and Peter Schüller (in alphabetical order), and to Herwig Schreiner for his comments on an earlier version of this paper.

1 Introduction

Answer Set Programming (ASP) is an approach to declarative problem solving [3, 22], in which problems to be solved by a computer are encoded as logic programs, which are sets of rules that can contain variables. Most ASP systems follow the *ground-and-solve* paradigm and split the solving process into two steps: First, a *grounder* transforms the input program containing variables into a propositional encoding [4, 12, 14, 19]. Then, solutions for the resulting variable-free program are found by a *solver* [16]. The grounding step can result in an exponential blow-up in space in the worst case [10].

This *grounding bottleneck* is a major problem of traditional approaches to ASP. For example, the rule

$$\mathit{partnerunits}(U,P) \leftarrow \mathit{unit2zone}(U,Z), \mathit{unit2sensor}(P,S), \mathit{zone2sensor}(Z,S), U \neq P.$$

has to be replaced by up to $|U| \cdot |P| \cdot |Z| \cdot |S|$ ground rules, where $|U|$, $|P|$, $|Z|$ and $|S|$ are the number of constants the respective variable may be substituted with. Many of these ground rules may be irrelevant because they are not needed to build a specific answer set anyway.

Problems that are actually easy to solve thus become prohibitive as soon as their grounding ceases to fit into working memory. This makes ASP, an otherwise powerful and versatile approach, unsuitable for large-scale problem instances frequently occurring in practice.

Lazy grounding interleaves grounding and solving to avoid storing the entire ground program in memory. By this, lazy grounding addresses the limitations of state-of-the-art grounders like GRINGO [14] and $\mathcal{I}$-DLV [4,12]. These grounders employ sophisticated grounding techniques to omit irrelevant ground rules, but these can only mitigate and not eliminate the blow-up in space. Known approaches to lazy grounding are ASPeRiX [20], GASP [25], OMiGA [7], and, most recently, Alpha [30]. Lazy grounding methods have also been proposed for FO($\cdot$), a knowledge representation formalism whose foundations are similar to those of ASP [8]. Also related, though not a lazy-grounding system as such, are *lazy constraints*, a technique that removes constraints that consume much space in grounding from the input program and adds only relevant ground ones again when a potential answer set has been found that needs to be checked for constraint violations [6].

While lazy-grounding systems are able to limit their memory usage, their time consumption is not yet 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 [30]. The system consists of a grounder and a solver which, however, do not work in sequence (as in ground-and-solve), but interact cyclically. Still, Alpha also does not 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.

The remainder of this research summary is structured as follows: In section 2, we recall preliminaries on ASP and lazy grounding. In section 3, we state the research questions addressed by this thesis, after which we report on its current status in section 4. Preliminary results are presented in section 5, before open issues and expected achievements are put forward in section 6. This paper is then briefly concluded in section 7.

2 Preliminaries

In this section, we present a brief account of ASP syntax and semantics and of the general idea of the lazy-grounding ASP solver Alpha.

2.1 Syntax

An answer-set program P is a finite set of rules of the form

$$h_1; \ldots; h_d \leftarrow b_1, \ldots, b_m, \mathit{not}\ b_{m+1}, \ldots, \mathit{not}\ b_n.$$

where $h_1, \ldots, h_d$ and $b_1, \ldots, b_m$ are positive literals (i.e. atoms) and $\mathit{not}\ b_{m+1}, \ldots, \mathit{not}\ b_n$ are negative literals. An atom is either a classical atom or a cardinality atom[1]. A classical atom is an expression $p(t_1, \ldots, t_n)$ where p is an n-ary predicate and $t_1, \ldots, t_n$ are terms. A term is either a variable or a constant. A literal is either an atom a or its default negation *not* a. Default negation refers to the absence of information, i.e. an atom is assumed to be false as long as it is not proven to be true. A *cardinality atom* is of the form

$$l\ \{a_1 : l_{1_1}, \ldots, l_{1_m}; \ldots; a_n : l_{n_1}, \ldots, l_{n_o}\}\ u$$

[1] Other types of atoms are supported in the language standard ASP-Core-2 [2], but these are not needed within the scope of this article.

where each structure $a_i : l_{i_1}, \ldots, l_{i_m}$ is a *conditional literal* in which a_i (the head of the conditional literal) and all l_{i_j} are classical literals, and l and u are terms representing non-negative integers indicating lower and upper bound. If one or both of the bounds are not given, their defaults are used, which are 0 for l and ∞ for u.

$H(r) = \{h_1, \ldots, h_d\}$ is called the *head*, and $B(r) = \{b_1, \ldots, b_m, \mathit{not}\ b_{m+1}, \ldots, \mathit{not}\ b_n\}$ the *body* of the rule. A rule r with $H(r)$ consisting of a cardinality atom is called *choice rule.* A rule with a head consisting of more than one classical atom is called *disjunctive rule.* A rule whose head consists of at most one classical atom is called a *normal rule.* A normal rule with empty head, e.g. $\leftarrow b.$, is called *(integrity) constraint.* A normal rule with empty body, e.g. $h \leftarrow .$, is called *fact.*

2.2 Semantics

There are several ways to define the semantics of an answer-set program, i.e. to define the set of answer sets of an answer-set program. An overview is provided by [23]. Probably the best-known semantics is based on the *Gelfond-Lifschitz reduct* [17]. A variant that applies to choice rules also is presented in [5].

Informally, an answer set A of a program P is a subset-minimal model of P (i.e. a set of atoms interpreted as *true*) which satisfies the following conditions: All rules in P are satisfied by A; and all atoms in A are “derivable” by rules in P. A rule is satisfied if its head is satisfied or its body is not. The disjunctive head of a rule is satisfied if at least one of its atoms is. A cardinality atom is satisfied if $l \leq |C| \leq u$ holds, where C is the set of head atoms in the cardinality literal whose conditions (e.g. $l_{i_1}, \ldots, l_{i_m}$ for a_i) are satisfied and which are satisfied themselves. In the presence of choice rules, the semantics is adjusted to allow non-minimal subsets satisfying the cardinality atom to appear in answer sets.

2.3 Lazy Grounding and Solving

ASP systems employing lazy grounding, such as Alpha, are based on so-called computation sequences, which are sequences of firing rules. Starting from facts, rules are fired one after the other by choosing in each step among the set of *applicable* rules, which are ground rules whose positive body is already satisfied and whose negative body is false or unassigned. This implies that the solver guesses whether an applicable rule fires, while traditional CDNL-based search guesses whether an arbitrary atom is *true* or *false.* Lazy-grounding solvers need an additional truth value *must-be-true* to distinguish whether an atom was derived by a firing rule or by a constraint [30]. When a conflicting assignment is reached, the solver backtracks. In this case, CDNL can learn new information from the conflict that is then used to avoid encountering similar conflicts in the future [16].

3 Research Questions

The remaining performance issues in lazy-grounding ASP solving lead us 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?

Within the scope of the DynaCon research project[2], lazy grounding methods will be evaluated on real-world industrial problem instances from domains like cyber-physical systems, road traffic control, and railway operation [11].

4 Current Status of the Research

To date, our research efforts have focused on research question 3, i.e. on heuristics. Preliminary results for domain-independent search heuristics have been reported at the 1st International Workshop on Practical Aspects of Answer Set Programming [28] and at the LPNMR 2017 Doctoral Consortium [27]. Lazy grounding and other ASP-based approaches to large-scale product configuration problems have been investigated in a contribution to the 19th International Configuration Workshop [26]. Since then, our research focus has shifted to domain-specific heuristics. A conference paper on this topic is currently being written.

5 Preliminary Results Accomplished

Source code has been contributed to Alpha, a lazy-grounding system introduced by Antonius Weinzierl [30]. 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 novel search heuristics.

5.1 Heuristics for Lazy-Grounding ASP Solving

Alpha takes ideas from state-of-the-art ASP solvers that work on a full grounding. Therefore it is natural to investigate heuristics from such systems and try to apply them in the solver component of a lazy-grounding system like Alpha. Heuristics for answer-set solving can roughly be classified into *domain-independent heuristics*, which are designed without a concrete application domain in mind, and *domain-specific heuristics*, which have to be tailored to a specific problem. For the class of domain-independent heuristics, two prominent examples are *VSIDS* [24] and *BerkMin* [18], which have originally been developed for SAT but are also successfully employed by ASP solvers (such as clasp [16] and wasp [1]). Both assign a so-called *activity* counter 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 the heuristic is asked for an atom, it chooses the most active unassigned atom. This is done to regard the fact that the set of variables responsible for conflicts may change very quickly. *BerkMin* additionally organizes the set of conflict clauses as a chronologically ordered stack, thereby preferring variables in recent conflicts. Other counters are maintained for picking truth values.

A direct application of *BerkMin* or *VSIDS* to a lazy-grounding ASP system 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 atoms, are known at any time to a lazy-grounding solver. Because of this, a heuristic that applies ideas from *BerkMin* or *VSIDS* to lazy grounding can

[2] Dynamic knowledge-based (re)configuration of cyber-physical systems, `https://isbi.aau.at/dynacon`

only incorporate atoms that are already grounded and thus 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 (cf. section 2.3).

5.2 Domain-Independent Heuristics

A set of domain-independent heuristics inspired by BerkMin and incorporating new ideas has been developed for Alpha and is described in detail in [28], where the results of a basic experimental evaluation on a number of benchmark problems can also be found. 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 was repeatedly able to outperform Alpha's naive heuristic as well as two BerkMin-inspired heuristics. It extends the latter by expanding the scope of atoms considered by the heuristics: If the atom a chosen due to its activity and recency is a choice point, i.e. it represents the body of an applicable rule, it is immediately picked. If that is not the case, the set of choice points *depending* on a are considered for selection, where a choice point representing the body of a ground rule $r\sigma$ is said to depend on all atoms occurring in $H(r\sigma) \cup B(r\sigma)$.

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.

5.3 Domain-Specific Heuristics

Domain-specific heuristics have been proposed for pre-grounding solvers but are not directly transferable to a lazy-grounding system for the same reasons that domain-independent heuristics are not. hclasp [15] is an extension of the solver clasp that accepts heuristic predicates as part of the declarative problem specification. These predicates allow to modify priorities and truth preferences of atoms, i.e. the order in which atoms are guessed and the truth values assigned to them during solving. These modifications can be mixed with domain-independent heuristics like VSIDS. Heuristic predicates have since been replaced by heuristic directives in clasp [13].

hwasp [9], on the other hand, is an extension of the solver wasp that facilitates the integration of external heuristics implemented in a procedural language which are consulted at specific points during the solving process via a prespecified API.

Our current efforts are directed at devising an extension of the ASP language by annotations or directives to specify heuristics declaratively within the input program. Our preliminary proposal for such annotations is syntactically similar to *optimize statements* in ASP-Core-2 [2] or *weak constraints* in DLV [21].

A rule to which an heuristic annotation is attached could be of the following form:

$$h_1; \dots ; h_d \leftarrow b_1, \dots , b_m, not\ b_{m+1}, \dots , not\ b_n.\ [w@l, s\colon c_1, \dots , c_o]$$

Here, w and l are terms denoting weight and level of the heuristics (together called *priority*, in which level is more important than weight), both defaulting to 1, and s is a sign (*true* or *false*) stating if the rule shall fire or not fire when selected.

During solving, all rules that have already been grounded and whose condition $c_1, \dots , c_o$ is satisfied by the current partial assignment[3] are candidates for rule selection. From these

[3] To be in line with semantics of default negation, atoms that are still unassigned are assumed false.

candidates, the solver will choose the one with the highest priority $w@l$ and assign s to the atom representing its body. If multiple applicable rules have the same maximum priority, a fallback heuristics like BerkMin or VSIDS is used to break ties. If s is not specified, the sign is also determined by a fallback heuristics. If a condition is not satisfied, the corresponding rule stays applicable but has default weight and level ($1@1$).

This syntax is still preliminary. We are currently working on extending it with ways to specify preferences over disjunctive heads or elements of a choice head, and with support for randomness and restarts. A full proposal together with example programs and a performance study will form a forthcoming publication.

6 Open Issues and Expected Achievements

So far, we have concentrated on search heuristics, i.e. on branching strategies for the solver component of a lazy-grounding ASP system. Several other procedures of such a system could be equipped with heuristic decision-making as well. A prominent group of such heuristics is formed by grounding heuristics, which 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($\cdot$) [8]. Ways to adapt and extend this for use by a CDNL-based solver will be explored.

Furthermore, we plan to explore other ways of modifying or enhancing the algorithms of a lazy-grounding ASP system to improve performance by aiding heuristic decision-making. For example, program analysis techniques (atom dependencies, strongly connected components, etc.) could guide the selection or tuning of solver heuristics, and relaxing the restrictions which atoms we can guess on could open up new ways towards finding answer sets quickly. By these efforts, we expect the solver heuristics to have more information and operate on a smaller search space – and thus to perform even better then currently is the case.

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 CDNL to generalise learnt nogoods to the non-ground level and reuse such non-ground nogoods for future problem instances. Preliminary work on non-ground rule learning has already been done in the scope of OMiGA [29]. There, a technique called *rule unfolding* is proposed, which is based on ideas from propositional resolution and derives new rules that are implied by the original program. A new rule that is a constraint containing variables can be seen as a non-ground nogood. To the best of our knowledge, non-ground rule learning has not yet been addressed in conjunction with CDNL, and the question how such learnt rules can be utilised to accelerate solving of future problem instances has also remained unstudied.

We expect that several aspects of our work will be beneficial not only for lazy-grounding systems, but for ground-and-solve systems as well.

7 Conclusion

Lazy grounding is an approach to solve the grounding bottleneck in ASP by interleaving grounding and solving. The performance of current lazy-grounding systems is not on par with that of systems producing the full grounding upfront, however. The goal of this thesis is to equip the lazy-grounding approach with novel heuristics and other techniques to make solving faster. Preliminary results have been accomplished in the area of domain-independent and domain-specific search heuristics. To the knowledge of the authors, this investigation of search heuristics in lazy-grounding ASP systems is the first of its kind. Future achievements on heuristics and conflict learning are expected to be transferable to ground-and-solve systems.

References

1 Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, pages 54–66. Springer, 2013. `doi:10.1007/978-3-642-40564-8_6`.

2 ASP Standardization Working Group. ASP-Core-2 input language format, 2012-12-13. URL: `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03b.pdf`.

3 Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011. `doi:10.1145/2043174.2043195`.

4 Francesco Calimeri, Davide Fuscà, Simona Perri, and Jessica Zangari. $\mathcal{I}$-dlv: The New Intelligent Grounder of dlv. In Giovanni Adorni, Stefano Cagnoni, Marco Gori, and Marco Maratea, editors, *AI*IA 2016: Advances in Artificial Intelligence - XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016, Proceedings*, volume 10037 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2016. `doi:10.1007/978-3-319-49130-1_15`.

5 Francesco Calimeri, Martin Gebser, Marco Maratea, and Francesco Ricca. Design and results of the Fifth Answer Set Programming Competition. *Artif. Intell.*, 231:151–181, 2016. `doi:10.1016/j.artint.2015.09.008`.

6 Bernardo Cuteri, Carmine Dodaro, Francesco Ricca, and Peter Schüller. Constraints, lazy constraints, or propagators in ASP solving: An empirical analysis. *TPLP*, 17(5-6):780–799, 2017. `doi:10.1017/S1471068417000254`.

7 Minh Dao-Tran, Thomas Eiter, Michael Fink, Gerald Weidinger, and Antonius Weinzierl. OMiGA : An open minded grounding on-the-fly answer set solver. In Luis Fariñas del Cerro, Andreas Herzig, and Jérôme Mengin, editors, *Logics in Artificial Intelligence - 13th European Conference, JELIA 2012, Toulouse, France, September 26-28, 2012. Proceedings*, volume 7519 of *Lecture Notes in Computer Science*, pages 480–483. Springer, 2012. `doi:10.1007/978-3-642-33353-8_38`.

8 Broes de Cat, Marc Denecker, Peter J. Stuckey, and Maurice Bruynooghe. Lazy Model Expansion: Interleaving Grounding with Search. *J. Artif. Intell. Res.*, 52:235–286, 2015. `doi:10.1613/jair.4591`.

9 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. `doi:10.1017/S1471068416000284`.

10 Thomas Eiter, Wolfgang Faber, Michael Fink, and Stefan Woltran. Complexity results for answer set programming with bounded predicate arities and implications. *Ann. Math. Artif. Intell.*, 51(2-4):123–165, 2007. `doi:10.1007/s10472-008-9086-5`.

11 Thomas Eiter, Gerhard Friedrich, Richard Taupe, and Antonius Weinzierl. Lazy Grounding for Dynamic Configuration. *KI - Künstliche Intelligenz*, May 2018. `doi:10.1007/s13218-018-0536-x`.

12 Wolfgang Faber, Nicola Leone, and Simona Perri. The Intelligent Grounder of DLV. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2012. `doi:10.1007/978-3-642-30743-0_17`.

13 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory Solving Made Easy with Clingo 5. In Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos, editors, *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016,*

New York City, USA, volume 52 of *OASICS*, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/OASIcs.ICLP.2016.2.

14 Martin Gebser, Roland Kaminski, Arne König, and Torsten Schaub. Advances in *gringo* Series 3. In James P. Delgrande and Wolfgang Faber, editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, pages 345–351. Springer, 2011. doi:10.1007/978-3-642-20895-9_39.

15 Martin Gebser, Benjamin Kaufmann, Ramón Otero, Javier Romero, Torsten Schaub, and Philipp Wanko. Domain-Specific Heuristics in Answer Set Programming. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, pages 350–356. AAAI Press, 2013. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6278.

16 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012. doi:10.1016/j.artint.2012.04.001.

17 Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

18 Evguenii I. Goldberg and Yakov Novikov. BerkMin: A Fast and Robust Sat-Solver. In *2002 Design, Automation and Test in Europe Conference and Exposition (DATE 2002), 4-8 March 2002, Paris, France*, pages 142–149. IEEE Computer Society, 2002. doi:10.1109/DATE.2002.998262.

19 Benjamin Kaufmann, Nicola Leone, Simona Perri, and Torsten Schaub. Grounding and Solving in Answer Set Programming. *AI Magazine*, 37(3):25–32, 2016. URL: http://www.aaai.org/ojs/index.php/aimagazine/article/view/2672.

20 Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia. ASPeRiX, a first-order forward chaining approach for answer set computing. *TPLP*, 17(3):266–310, 2017. doi:10.1017/S1471068416000569.

21 Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006. doi:10.1145/1149114.1149117.

22 Vladimir Lifschitz. What Is Answer Set Programming? In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1594–1597. AAAI Press, 2008. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-270.php.

23 Vladimir Lifschitz. Thirteen Definitions of a Stable Model. In Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *Lecture Notes in Computer Science*, pages 488–503. Springer, 2010. doi:10.1007/978-3-642-15025-8_24.

24 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. doi:10.1145/378239.379017.

25 Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. GASP: answer set programming with lazy grounding. *Fundam. Inform.*, 96(3):297–322, 2009. doi:10.3233/FI-2009-180.

26 Gottfried Schenner and Richard Taupe. Techniques for Solving Large-Scale Product Configuration Problems with ASP. In Linda L. Zhang and Albert Haag, editors, *Proceedings of the 19th International Configuration Workshop*, pages 12–19, La Défense, France, 2017. URL: `https://www.ieseg.fr/wp-content/uploads/2017/01/Proceedgins_FinalV2.pdf#page=12`.

27 Richard Taupe. Lazy Grounding and Heuristic Solving in Answer Set Programming. In Marina de Vos, editor, *LPNMR 2017 Doctoral Consortium*, Bath, 2017. URL: `http://lpnmr-dc.cs.bath.ac.uk/programme/Proceedings.pdf#page=41`.

28 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. URL: `https://sites.google.com/site/paoasp2017/Taupe-et-al.pdf`.

29 Antonius Weinzierl. Learning Non-Ground Rules for Answer-Set Solving. In David Pearce, Shahab Tasharrofi, Evgenia Ternovska, and Concepción Vidal, editors, *2nd Workshop on Grounding and Transformations for Theories With Variables*, pages 25–37, 2013. URL: `http://kr.irlab.org/sites/10.56.35.200.gttv13/files/gttv13.pdf#page=31`.

30 Antonius Weinzierl. Blending Lazy-Grounding and CDNL Search for Answer-Set Solving. In Marcello Balduccini and Tomi Janhunen, editors, *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2017. `doi:10.1007/978-3-319-61660-5_17`.