Design and Implementation of the $\mathcal{I}$-*DLV* Grounder: Optimizations, Customizability and Interoperability

Jessica Zangari

Department of Mathematics and Computer Science, University of Calabria, Rende, Italy
zangari@mat.unical.it

Abstract. $\mathcal{I}$-*DLV* is the result of an ex-novo design and implementation of a new modern and efficient ASP instantiator. It features full support to ASP-Core-2 standard language, deductive database capabilities, increased flexibility and an extensible design that eases the incorporation of optimization techniques, language updates and customizability. Recently, the system has been endowed with means aimed at easing interoperability and integration with external systems, and also at accommodating external source of computation and value invention within ASP programs. Moreover, its usage is twofold: besides being an efficient stand-alone grounder, it is also a full-fledged deductive database engine. Along with the solver *WASP* it has been integrated in the new version of the widespread ASP system *DLV* recently released.

Keywords: Artificial Intelligence, Knowledge Representation and Reasoning, Answer Set Programming, Grounding, Instantiation, Deductive Database Systems, *DLV*

1 Introduction

Answer Set Programming (ASP) [15, 16] is a purely declarative logic formalism developed in the field of logic programming and non-monotonic reasoning. ASP became widely used in AI and recognized as a powerful tool for Knowledge Representation and Reasoning. The basic construct of ASP is a rule, that has form $Head \leftarrow Body$, where the $Body$ is a logic conjunction in which negation may appear, and $Head$ can be either an atomic formula or a logic disjunction. A rule is interpreted according to common sense principles: roughly, its intuitive semantics corresponds to an implication. In ASP a problem is modeled via a logic program composed by a collection of rules; an ASP system is in charge of determining its solutions by computing its intended models, called *answer sets*, according to the so called *answer set semantics*. Answer sets correspond one-to-one to a solution of the modeled problem; if a program has no answer sets, the corresponding encoded problem has no solutions. ASP roots stem from Datalog [10], a popular declarative logic programming language also based on rules; however, differently from Datalog, ASP may admit disjunctive heads and non-stratified negation. These distinguishing features make ASP suitable for modelling non-monotonic reasoning. Theoretically, the introduction of disjunction in rule heads yields to a more expressive paradigm allowing to capture the complexity class $\Sigma_2^P = \mathrm{NP}^{\mathrm{NP}}$.

Throughout the years a significant effort has been spent in order to extend the "basic" language and ease knowledge representation tasks with ASP; it has been proven to be highly versatile, offering several language constructs and reasoning modes. Recently, the community agreed on a standard input language for ASP systems: ASP-Core-2, the official language of the ASP Competition series [14]. Furthermore, ASP has been successful adopted for developing advanced applications in many research areas, ranging from Artificial Intelligence to Databases and Bioinformatics, as well as in industrial contexts [20].

The "traditional" approach to the evaluation of ASP programs relies on a grounding module (*grounder*), that generates a propositional theory semantically equivalent to the input program, coupled with a subsequent module (*solver*) that applies propositional techniques for generating its answer sets [17]. There have been other attempts deviating from this customary approach [11, 18, 19, 22]; nonetheless, the majority of the current solutions relies on the canonical "ground & solve" strategy, as systems relying on such approach proved to be more reliable and high-performance in the widest range of scenarios, to date.

After more than twenty years of research the theoretical properties of ASP are understood, while the linguistic extensions introduced with ASP-Core-2, their effects on the expressive power of ASP, and the ASP-based applications arising from a broader range of scenarios demand for increasingly high-performance implementations. In addition, while the solving phase has been more largely investigated in literature [21], less emphasis has been placed on the instantiation phase. Nevertheless, the grounding solves a complex problem, which is in general EXPTIME-hard: the produced ground program is potentially of exponential size with respect to the input program [12]. Grounding, hence, may be computationally expensive and has a big impact on the performance of the whole system, as its output is the input for the subsequent solving step, that, in the worst case, takes exponential time in the size of the input [3, 4].

2 The New Grounder $\mathcal{I}$-*DLV*

In our work, we addressed the existing grounding techniques with the aim of improving them and then introducing novel optimizations; on such bases we consequently designed and developed a new instantiator for ASP, namely $\mathcal{I}$-*DLV*. We started by improving some already known techniques having a high impact on the overall instantiation process; in addition, we designed a set of fine-tuning optimizations acting to different extents on the instantiation process, with the general common aim of reducing the search space and improving overall performance [5]. Furthermore, based on the long-lasting experience from the ASP competition series [14], given that the same computational problem can be encoded by means of many different ASP programs which are semantically equivalent, we noticed as ASP systems may perform very differently when evaluating each one of them. This issue, in a certain sense, conflicts with the declarative nature of ASP, that should free the users from the burden of the computational aspects. Therefore, we defined a heuristic-guided technique to transform an ASP program into an equivalent one that can be evaluated more efficiently [8].

The introduced grounding strategies lend themselves to the integration in a generic grounder module of a traditional ASP system following a ground & solve approach. In particular, the novel system *$\mathcal{I}$-DLV* incorporates all of them leveraging on their synergy to perform an efficient instantiation.

$\mathcal{I}$-DLV has been redesigned and re-engineered from scratch. Differently from *DLV*, *$\mathcal{I}$-DLV* natively supports ASP-Core-2 and it is compatible with state-of-the-art technologies. The foremost issue experienced is the high-influence of grounding on solving; in general, simply improving the grounding times does not necessary imply improvements on the solving side, since these heavily depend on the structure and form of the produced instantiation. Thus, *$\mathcal{I}$-DLV* grounding process has been endowed with high flexibility and customizability, thanks to a lightweight modular design that eases the incorporation of optimization techniques and future updates. In particular, one of the novelty is the customizable nature of the grounder, allowing to tailor its produced instantiation to different extents, such as to better conform to solvers needs and to experiment with ASP and its applications for better adapting ASP-based solutions to real-world applications. The novel possibility of *annotating ASP code* with external directives to the grounder is a bold move in this direction, providing a new way for fine-tuning both ASP programs and systems for any specific scenario at hand [7].

Despite being released recently, *$\mathcal{I}$-DLV* performance is promising and comparable with mainstream systems: in the latest ASP Competition [14] *$\mathcal{I}$-DLV* ranked both the first and second positions when combined, respectively, with an automatic solver selector [6] that inductively chooses the best solver depending on some inherent features of the instantiation produced, and with the state-of-the-art solver *clasp* [13]. Moreover, *$\mathcal{I}$-DLV* is an open-source project: its source and binaries are available from the official repository [9].

Eventually, the system is envisioned as core part of a larger project comprising the extension of *$\mathcal{I}$-DLV* towards mechanisms for interoperability with other formalisms and tools [7]. The intent is to foster the usage of ASP, and in general, of logic programming in real-world and complex applications.

3 Customizability and Interoperability

As briefly introduced before, the input language of *$\mathcal{I}$-DLV* has been enriched with the support for special comments expressing meta-data information that *$\mathcal{I}$-DLV* can interpret in order to fine-tune its grounding process. Following a widespread term in programming, we named these constructs *annotations*, as they do not change the semantics of input programs, but their impact might be observed just on the performance. For this reason, their notation starts with

%, which is used for comments in ASP-Core-2, so that other systems can simply ignore them. In *$\mathcal{I}$-DLV*, annotations allow to give explicit directions on the internal computational process. In particular, supported annotations belong to the following categories: *grounding annotations* allowing for a fine-grained customization on the grounding process, and *solving annotations* that have been integrated into *DLV2*, and are geared to the customization of the whole computational process [1].

As additional features, *$\mathcal{I}$-DLV* has been endowed with means to ease the interoperability with external sources of knowledge. In particular, *$\mathcal{I}$-DLV* now supports calls to Python scripts via *external atoms*, and connection with relational and graph databases via explicit *directives* for importing/exporting data.

For further insights about customizability and interoperability in *$\mathcal{I}$-DLV* we refer the reader to [7].

4 Conclusion and Future Work

The *$\mathcal{I}$-DLV* project is under active development; further enhancements are planned related to language extensions, customizability means, performance, and a tight integration with ASP solvers.

In particular, we plan to extend the set of directives for interoperating with external data and further enlarge the set of available annotations. In addition, we are studying a proper way for manipulating the produced ground program in order to better fit with the computational mechanisms carried out by ASP solvers, and we plan to endow *$\mathcal{I}$-DLV* with further pre-processing steps aiming at making performance less encoding-dependent: we believe that such means are of great importance for fostering and easing the usage of ASP in practice, fully complying with the declarative power of ASP.

References

1. Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P., Zangari, J.: The ASP system DLV2. In: Balduccini and Janhunen [2], pp. 215–221
2. Balduccini, M., Janhunen, T. (eds.): Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings, LNCS, vol. 10377. Springer (2017)
3. Ben-Eliyahu, R., Dechter, R.: Propositional Semantics for Disjunctive Logic Programs. Annals of Mathematics and Artificial Intelligence 12, 53–87 (1994)
4. Ben-Eliyahu, R., Palopoli, L.: Reasoning with Minimal Models: Efficient Algorithms and Applications. In: Proceedings Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94). pp. 39–50 (1994)
5. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: *$\mathcal{I}$-DLV*: The New Intelligent Grounder of DLV. Intelligenza Artificiale 11(1), 5–20 (2017)
6. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: *$\mathcal{I}$-DLV+$\mathcal{MS}$*: Preliminary Report on an Automatic ASP Solver Selector. In: 24th RCRA International Workshop on "Experimental Evaluation of Algorithms for solving problems with combinatorial explosion" (2017), To appear

7. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: External Computations and Interoperability in the new DLV Grounder. In: Proceedings of the 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017). Bari, Italy (2017), To appear
8. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: Optimizing Answer Set Computation via Heuristic-Based Decomposition. In: Practical Aspects of Declarative Languages - 19th International Symposium, PADL 2018, Los Angeles, CA, USA, 8-9 January 2018, Proceedings (2018), To appear
9. Calimeri, F., Perri, S., Fuscà, D., Zangari, J.: $\mathcal{I}$-DLV repository (since 2016), https://github.com/DeMaCS-UNICAL/I-DLV
10. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about Datalog (and never dared to ask). IEEE Transactions on Knowledge and Data Engineering 1(1), 146–166 (1989)
11. Dal Palù, A., Dovier, A., Pontelli, E., Rossi, G.: GASP: Answer Set Programming with Lazy Grounding. Fundamenta Informaticae 96(3), 297–322 (2009)
12. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys 33(3), 374–425 (2001)
13. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artificial Intelligence 187-188, 52–89 (2012)
14. Gebser, M., Maratea, M., Ricca, F.: The design of the seventh answer set programming competition. In: International Conference on Logic Programming and Nonmonotonic Reasoning. pp. 3–9. Springer (2017)
15. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, WA, Aug 15-19, 1988 (2 Volumes). pp. 1070–1080. MIT Press, Cambridge, Mass. (1988)
16. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)
17. Kaufmann, B., Leone, N., Perri, S., Schaub, T.: Grounding and solving in answer set programming. AI Magazine 37(3), 25–32 (2016)
18. Lefèvre, C., Nicolas, P.: A first order forward chaining approach for answer set computing. In: Erdem, E., Lin, F., Schaub, T. (eds.) Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings. LNCS, vol. 5753, pp. 196–208. Springer (2009)
19. Lefèvre, C., Nicolas, P.: The first version of a new asp solver : Asperix. In: Erdem, E., Lin, F., Schaub, T. (eds.) Logic Programming and Nonmonotonic Reasoning – 10th International Conference (LPNMR 2009). LNCS, vol. 5753, pp. 522–527. Springer Verlag (Sep 2009)
20. Leone, N., Ricca, F.: Answer set programming: A tour from the basics to advanced development tools and industrial applications. In: Faber, W., Paschke, A. (eds.) Reasoning Web. Web Logic Rules - 11th Int'l Summer School 2015, Berlin, Germany, Jul 31 - Aug 4, 2015, Tutorial Lectures. LNCS, vol. 9203, pp. 308–326. Springer (2015)
21. Lierler, Y., Truszczynski, M.: Transition systems for model generators - A unifying approach. Theory and Practice of Logic Programming 11(4-5), 629–646 (2011)
22. Weinzierl, A.: Blending lazy-grounding and CDNL search for answer-set solving. In: Balduccini and Janhunen [2], pp. 191–204