Incremental Lazy Grounding (Work in Progress)

Sebastian Adam[1][0009–0000–9905–0128] and Thomas Eiter[1][0000–0001–6003–6345]

Technische Universität Wien, Vienna, Austria
`sebastian.adam,thomas.eiter@tuwien.ac.at`

Abstract. We introduce an incremental extension of the lazy grounding answer set solver Alpha. Unlike traditional systems such as iclingo and I-DLV, which combine incremental solving with full grounding, our approach retains the advantages of lazy grounding (viz., reduced grounding effort) and enables multi-shot solving for problems where the ground-and-solve approach is infeasible. A Python prototype shows reduced grounding times and hence faster solving on dynamic 3-coloring benchmarks. This establishes the first step toward integrating incremental solving into lazy grounding, addressing scalability in evolving domains.

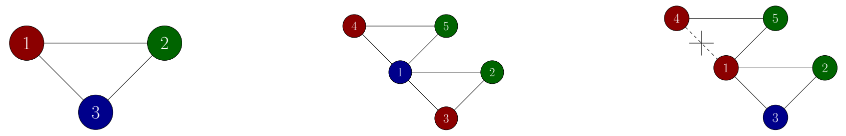
Motivation. Incremental solving enhances answer set programming (ASP) by reusing information across multiple solving shots. This is useful in domains such as planning, where problems naturally evolve over time, and for bounded reasoning tasks, where incremental solving offers an intuitive way to increase the bound.

State-of-the-art ASP solvers like clingo [6] and DLV [1] support this multi-shot reasoning via the frameworks iclingo [5] (now integrated in clingo proper [7]) and I-DLV [2], respectively. However, they rely on the traditional "ground-and-solve" approach. In contrast, lazy grounding solvers such as Alpha [8] avoid grounding the full program upfront and only ground applicable rules on demand during search. This makes them suitable for domains that suffer from the grounding bottleneck [4, 3] (i.e., an exponential blow-up in grounding).

Despite their incremental grounding approach, no lazy grounding solver currently supports incremental solving akin to iclingo or I-DLV. Although the incremental solving paradigm aligns well with lazy grounding, integrating the two is non-trivial. Many techniques used by state-of-the-art traditional ASP solvers assume a full grounding and hence are not applicable in a lazy grounding setting. To address this, we introduce a novel algorithm that extends the core loop of the lazy grounder Alpha for the incremental solving setting.

Theory. The core algorithm of Alpha integrates grounding and solving in a loop, similar to a CDNL solver [8]: applicable rules are grounded and turned into nogoods $\rightarrow$ propagation ensures consistent literal extension $\rightarrow$ if a conflict is detected, perform conflict analysis, learn a nogood, and backjump. This loop repeats until all literals are assigned a truth value, representing an answer set.

Traditionally, after execution, grounded rules and learned nogoods are dismissed. In an incremental setting, with multiple solving shots that add or remove



(a) Initial Configuration (b) Add *node*(4..5), *edge*(1, 4), *edge*(4, 5), *edge*(5, 1) (c) Remove *edge*(1, 4)

Fig. 1: 3-coloring example with changing structure over multiple solving shots

facts, it is however beneficial to retain as much information as possible. Unfortunately, maintaining the entire state of the lazy grounding solver after a solving shot is not feasible. Among other things, this is because saved choices in the search tree and enumeration nogoods (i.e., nogoods that prevent recomputing an answer set) hinder exploration of the full search space.

To address this issues, we classify nogoods created in each solving shot as representing (1) facts, (2) grounded rules, (3) an answer set (enumeration), or (4) combined information from multiple nogoods (resolution). Enumeration and resolution nogoods are then deleted between shots, and choices made by the solver are reset. If a fact is removed, its corresponding nogood is eliminated. Remaining fact and rule nogoods created by the grounder are still applicable and can therefore be reused in the next solving shot, reducing grounding effort.

Implementation. For implementing our incremental extension of Alpha, we re-implemented its core algorithm in Python. This version focuses on key steps of the algorithm and omits involved heuristics and watched literals used by the full Java implementation. Thus, while performance of the original solver is far superior, we gain greater extendibility for rapid prototyping.

To illustrate the capabilities of our incremental lazy grounding solver, we compare it against the static version of the same Python implementation. Both versions use the enumeration mode (i.e., compute all answer sets) and average the results over 100 runs. As a benchmark, we use the 3-coloring example from [2] shown in Figure 1. In the initial configuration (Figure 1a), both versions perform equally, grounding the instance in 7 ms. After adding new facts (Figure 1b), grounding time rises to 46 ms for static and 37 ms for incremental, showing the performance gain from reusing grounding information. In the third shot (Figure 1c), where an edge is removed, the incremental version requires no extra grounding, while the static spends 37 ms grounding.

For a more comprehensive performance analysis, we plan to implement the incremental extension in the full Java version of Alpha and test different problems. Further work could also investigate retaining resolution nogoods — currently, all are removed between shots, though some might still apply. Moreover, instead of focusing only on grounding, we could explore equipping the solver with knowledge of learned answer sets, heuristics, etc. to speed up solving, similar to iclingo [5].

Acknowledgments. This project is part of the TrustACPS Doctoral College at TU Wien and supported by Vienna Science and Technology Fund (WWTF) project ICT22-023.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

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: LPNMR. Lecture Notes in Computer Science, vol. 10377, pp. 215–221. Springer (2017)
2. Calimeri, F., Ianni, G., Pacenza, F., Perri, S., Zangari, J.: Asp-based multi-shot reasoning via DLV2 with incremental grounding. Theory Pract. Log. Program. **25**(3), 281–303 (2025)
3. Cuteri, B., Dodaro, C., Ricca, F., Schüller, P.: Overcoming the grounding bottleneck due to constraints in ASP solving: Constraints become propagators. In: IJCAI. pp. 1688–1694. ijcai.org (2020)
4. Eiter, T., Faber, W., Fink, M., Woltran, S.: Complexity Results for Answer Set Programming with Bounded Predicate Arities. Annals of Mathematics and Artificial Intelligence **51**(2-4), 123–165 (2007)
5. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In: ICLP. Lecture Notes in Computer Science, vol. 5366, pp. 190–205. Springer (2008)
6. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = ASP + control: Preliminary report. CoRR **abs/1405.3694** (2014)
7. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory Pract. Log. Program. **19**(1), 27–82 (2019)
8. Leutgeb, L., Weinzierl, A.: Techniques for efficient lazy-grounding ASP solving. In: DECLARE. Lecture Notes in Computer Science, vol. 10997, pp. 132–148. Springer (2017)