Omission-based Abstraction for Answer Set Programs* (Extended Abstract)

Zeynep G. Saribatur and Thomas Eiter

Institute of Logic and Computation, TU Wien

Abstraction is a widely used approach in computing solutions for hard problems by over-approximating them. By a deliberate loss of information, the problem is approximated to achieve a smaller or simpler state space, at the price of spurious counterexamples to the behavior. The well-known counterexample guided abstraction refinement (CEGAR) [4] is based on starting with an initial abstraction on a given program and checking the desired property over the abstract program. Upon encountering spurious solutions, the abstraction is refined by removing the spurious transitions observed through the solution, so that the spurious solution is eliminated from the abstraction. This iteration continues until a concrete solution is found.

In this paper, we make the first step towards employing the concept of abstraction in ASP. We are focused on abstraction by omitting atoms from the program and constructing an abstract program with the smaller vocabulary that preserves the original structure of the rules, by ensuring that the original program is over-approximated, i.e., every original answer set can be mapped to some abstract answer set. Due to the decreased search size, finding an answer set in the abstract program is easier, while one needs to check whether the found abstract answer set is concrete. As spurious answer sets can be introduced, one may need to go over all abstract answer sets until a concrete one is found. If the original program has no answer set, all encountered abstract answer sets will be spurious. To eliminate spurious answer sets, we use a CEGAR inspired approach, by finding a cause of the spuriousness with ASP debugging [3] and refining the abstraction by adding back some atoms that are deemed to be "badly-omitted".

An interesting application area for such an omission-based abstraction in ASP is finding an *explanation* for unsatisfiability of programs. Towards this problem, debugging inconsistent ASP programs has been investigated [3, 9, 5, 6], which is based on providing the reason (i.e., occurring violations) on why an expected solution provided by the user does not exist. However, these methods do not address the question of why the program does not give any solutions. We approach the unsatisfiability of an ASP program differently with an interest in obtaining a projection of the program which shows the cause of the unsatisfiability, without an initial idea on expected solutions. The well-known notion of minimal unsatisfiable subsets (*unsatisfiable cores*) [7, 8] has also been used in the ASP context [1, 2], and we discuss the relation to the spurious answer sets.

Our contributions are briefly summarized as follows.

* Appears in the Proceedings of KR 2018, pages 42–51.

- We introduce a method to automatically abstract ground ASP programs Π by omitting atoms in order to obtain an over-approximation of the answer sets of Π. That is, a program Π' is constructed such that each answer set I of Π is abstracted to some answer set I' of Π'. While this abstraction is many to one, *spurious* answer sets of Π' may exist that do not correspond to any answer set of Π.
- We present a refinement method inspired by ASP debugging approaches to catch badly omitted atoms through the encountered spurious answer sets.
- We introduce the notion of *blocker set* as a set of atoms such that abstraction to it preserves unsatisfiability of a program. A minimal blocker set then gives a projection of the program to the minimal cause of unsatisfiability.
- We derive complexity results for the notions, such as for checking for spurious answer sets, finding minimal sets of atoms to put back in the refinement to eliminate a spurious solution, and computing a minimal blocker for a program.
- We report about preliminary experiments[1] focusing on unsatisfiable programs and investigate computing minimal blockers of programs. We compare the results of the abstraction and refinement approach starting with an initial abstraction (*bottom-up*) with a naive *top-down* approach that omits atoms one-by-one if their omission preserves unsatisfiability, and we observe that the bottom-up approach can obtain smaller sized blockers.

Overall, abstraction by omission appears to be of interest for ASP, which besides explaining unsatisfiability can be utilized, among other applications, to over-approximate reasoning and to represent projected answer sets.

References

1. Alviano, M., Dodaro, C.: Anytime answer set optimization via unsatisfiable core shrinking. Theory and Practice of Logic Programming **16**(5-6), 533–551 (2016)
2. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: Proc. ICLP. vol. 17, pp. 211–221 (2012)
3. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: Debugging asp programs by means of asp. In: Proc. LPNMR. pp. 31–43. Springer (2007)
4. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. JACM **50**(5), 752–794 (2003)
5. Dodaro, C., Gasteiger, P., Musitsch, B., Ricca, F., Shchekotykhin, K.: Interactive debugging of non-ground asp programs. In: Proc. LPNMR. pp. 279–293 (2015)
6. Gebser, M., Pührer, J., Schaub, T., Tompits, H.: A meta-programming technique for debugging answer-set programs. In: Proc. AAAI. vol. 8, pp. 448–453 (2008)
7. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning **40**(1), 1–33 (2008)
8. Lynce, I., Silva, J.P.M.: On computing minimum unsatisfiable cores. In: Proc. SAT (2004)
9. Oetsch, J., Pührer, J., Tompits, H.: Catching the ouroboros: On debugging non-ground answer-set programs. TPLP **10**(4-6), 513–529 (2010)

[1] The tool is available at `www.kr.tuwien.ac.at/research/systems/abstraction`.