Answer Set Programming: The Grounding and Solving Process

Marvin Beese, mabeese@uni-potsdam.de, 786300

University of Potsdam

Abstract. Answer Set Programming is a declarative approach for solving problems using logic programs. In the solution process, a grounder and a solver are used for the effectively instantiation of variables respectively for computing an answer set. In this summary, I want to explain important parts of the grounding and solving process.

1 Introduction

This summary is based on the article "Grounding and Solving in Answer Set Programming" by Benjamin Kaufmann, Nicola Leone, Simona Perri and Torsten Schaub [1]. In Answer Set Programming (ASP) a problem is modeled as a logic program. The grounder replaces all variables in a program and on that basis, the solver computes an answer set. In the following, I am going to describe the grounding process and the solving process. This includes on the one hand the iterative rule-instantiation procedure over body literals, and the optimization process with the dynamic magic set technique, and on the other hand unit propagation.

2 The Grounding Process: Instantiation and Optimization

In Answer Set Programming a grounder is used to generate a ground program, which is a program that does not contain any variable. By grounding, complex programs become simpler, which has huge consequences for the solver, because it expects the simplified program as an input for solving. For efficiently producing a ground program, the grounder uses smart procedures, which output a smaller ground program than with the full instantiation, where also useless rules are included. The semantics on the other hand are still preserved.

The instantiation procedure generates ground instances of a rule for given ground atoms. Iteratively, possible substitutions of the variables of body literals are searched. This is done under consideration of the safety condition that each rule variable also has to appear in a positive body literal. Rules are instantiated respectively of matching ground atoms. If the grounder finds one solution, a backtracking step for finding other possible rule instances happens, which restores the assignment of a variable and starts the search for a new match. The

instantiation over body literals takes place from left to right. There exists an instance for a rule, if all body literals of a rule have been instantiated. Facts are a set of ground atoms, which establish the starting point of the computation. We consider a set of extensions S, where initially $S = \text{Facts}$. During instantiation, we expand S with the ground atoms of newly generated ground rules. It is important to follow the evaluation order of rules. For example, if a rule r_1 has a specific predicate in its head and another rule r_2 contains the same predicate in its body, then r_1 has to be evaluated before r_2. An anomaly are recursive rules, where a body predicate depends on the predicate in the head of a rule. Here several iterations are performed until a fixpoint is reached and to avoid duplicates, only newly generated ground atoms are taken into account at each iteration.

There are some improvements of the instantiation process. For example, the grounder uses the magic set technique, where the input program is rewritten so that only a relevant subset of the program is used for answering the query. The magic predicates are extensions that represent relevant atoms respecting the query. For disjunctive programs, dynamic magic set techniques are used, which are able to exploit the information provided by magic predicates also during the answer set search. This has applications also on some co-NP complete problems (Manna, Ricca, and Terracina 2015). There exist also other optimization techniques like backjumping, which preserves the programs semantics but reduces the size of the ground program and parallel instantiation, which benefits of multicore and multiprocessor computers.

3 The Solving Process

The solving process is based on conflict driven search procedures and relies on the stable model's semantics, which requires that atoms are provably true. With disjunctive rules and nonmonotone aggregates, there exists an elevated level of complexity, which results in additional search efforts. Because various reasoning modes are possible, there are challenges regarding flexible solver architectures and dealing with inferences.
In the unit propagation, the solver interprets an ASP program into propositional logic. This can be read as a set of implications, where multiple body literals within one rule are a conjunctive implication and body literals of different rules but the same head are summarized into a disjunctive implication. After this happens the elimination of models with unsupported atoms, so that the remaining implications result in equivalences. It is also important to determine, whether the derivation of circular rules is harmful. Therefore, the solver checks the existence of valid external support by providing a rule whose head is in the loop but none of its positive antecedents belongs to it. The satisfaction of this rule can be tested in linear time. To translate a program into nogoods, the solver works the same way but also considers the space issue.

4 Conclusion

In Answer Set Programming, grounding benefits from efficient instantiation and evaluation processes, which apply optimization too. The solver contains search procedures and effective unit propagation. Powerful grounding and solving techniques are possible within a high-level modeling language, where these engines can be used as black-box systems.

References

1. Benjamin Kaufmann, Nicola Leone, Simona Perri, Torsten Schaub: Grounding and Solving in Answer Set Programming. AI Magazine 37(3): 25-32 (2016)