Grounding Paradigms in the Development of ASP Applications

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

University of Potsdam

Abstract. Answer Set Programming (ASP) can be successfully used in industrial applications. This enables new ways of computing challenging problems, but also requires development processes and designs for appropriate application and integration. This summary examines these processes and designs as well as grounding paradigms to solve large problems efficiently.

1 Introduction

This summary is based on the article "Industrial Applications of Answer Set Programming" by A. Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, Erich Christian Teppan [1]. One of the main fields of AI is automatic generation of solutions, which can be implemented declaratively with Answer Set Programming. This has many benefits for industry applications, as it can reduce the implementation and maintenance costs and user interaction. In the development process of ASP applications exist different methodologies, such as the guess-and-check or achievement-based methodology. Taking that and industry standards under consideration, there are several development steps for ASP applications, that I will consider in this summary. Later I will look into the grounding paradigms ground-and-solve, constraint asp, multi-shot solving and lazy grounding.

2 The Development Process and Design of ASP Applications

A common technique for ASP encoding is the guess-and-check methodology, where solution candidates are generated and invalid ones will be deleted with the use of constraints. In contrast to this exists the more proof-driven approach 'achievement-based' ASP, where written rules will be validated regarding their correctness. The latter lacks support of default rules.
Several steps for the development are manifested, with taking these methodologies and further project management standards for industrial applications into account. After the requirement for ASP applications is identified, an ASP specification of the core problem must be implemented and validated against small test instances. In the performance engineering step, there must be tests of

scalability for industrial use and comparisons of alternative encodings. As ASP-programs normally act as an extension of already existing software, the program must be implemented in the existing environment, tested and debugged against automated regression tests and maintained, for example with a documentation, which simplifies the possibility of later extensions or adaptions.
Industrial applications are mostly modelled in an object-oriented paradigm (OO), where modeling-languages like UML specify the classification via a class diagram. In the design of ASP applications, the analysis of OO software follows an OOASP approach, where an OOASP program encodes classes, attributes, associations and constraints with facts and derives relations between object-instances from a modeling-language. I will not go into detail of supported reasoning tasks, as this would get more technical.

3 Grounding Paradigms in ASP

A common paradigm for ASP systems is the ground-and-solve paradigm, where the grounder constructs a propositional encoding for a given input program, which will be solved in the second step by the solver. Because the grounding step results in an increased size of the program, ASP programs should have an implementation to handle large problem instances in grounding. If no program exists for the required size of problem instances, this leads to the so called 'grounding bottleneck'.
Another paradigm is the constraint ASP (CASP), which is a hybrid of ASP and constraint programming. There are two variants of CASP, either ASP gets extended for the formulation of constraints, or ASP acts as a specification language for constraint solving problems, where answer sets function as encodings for the constraint solver.
In multi-shot answer set solving, logic programs continously change. Thereby, an expansion of the domain happens with every changing step, until the program finds a solution. As the domain extends only for the problem-required size, this enables the finding of the domain size while searching for a solution and additionally, the ground program is not as large as it can get, rather it only stays as small as it is needed to be.
To enable a paradigm for tackling custom ASP encodings, lazy grounding requires only standard ASP encodings. Lazy encoding deals with the grounding bottleneck by composing grounding and searching and benefits with the limitation of memory usage. On the other hand, leads the lack of conflict-driven nogood learning of most existing solvers to incomparable time-consumption with other paradigms.

4 Conclusion

The use of ASP for industrial applications opens new ways for solving of intractable or difficult problems, like planning, scheduling or timetabling. But ASP has also applications in Configuration, Software Engineering and many

more fields, mainly because of its reduction of implementation and maintenance costs. To meet the requirements of industrial regulations and also different encoding paradigms of ASP, development steps exist for the development process and application design. This includes the identification of needs and valid specification of problems as well as performance comparisons and integration into existing systems. Further, object oriented software can be analyzed with the use of OOASP.
Grounding in ASP can effect in problems regarding the problem bottleneck. With the introduction of CASP, multi-shot solving and lazy grounding, this problem can be tackled.

References

1. Andreas A. Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, Erich Christian Teppan: Industrial Applications of Answer Set Programming. KI 32(2-3): 165-176 (2018)