Knowledge Representation and Reasoning about Effects of Actions

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

University of Potsdam

Abstract. Answer Set Programming is declarative programming paradigm with various application fields. With the use of knowledge bases and exceptions to defaults reasoning gets more powerful. ASP is also capable of computing effects of actions and to outsource external computations. I will discuss knowledge reasoning and hybrid reasoning, effects of actions and the advantages of Answer Set Programming.

1 Introduction

Answer Set Programming (ASP) is a declarative paradigm for knowledge representation and reasoning due to its expressivity with recursive definitions, aggregates, weight constraints, optimization statements (Lifschitz 2016), default negation and external atoms. ASP has main fields of applications both in research and in industry in robotics, bioinformatics, computational biology, diagnosis, automatic music composition, linguistics with natural language processing and understanding and more. In the first part of this summary, I will show the representation of knowledge and describe nonmonotonic reasoning with defaults. In the second part, I will be talking about representing a changing domain and effects of actions, hybrid reasoning using external atoms and advantages of ASP compared to imperative languages. This summary is of the paper "Applications of Answer Set Programming" by Esra Erdem, Michael Gelfond and Nicola Leone [1].

2 Knowledge Representation and Nonmonotonic Reasoning

In the field of computational intelligence, the reasoning of agents in a changing environment is one of the main goals. To describe the mathematical and nonmathematical model of an environment and the capabilities and goals of an agent, a knowledge base (KB) is used. A KB consists of commonsense knowledge, represented by nonmathematical models, and specialized knowledge, represented by a set of activities. For representing commonsense knowledge, describing non-changing environments uses monotonic reasoning with the rule, that a statement stays proven, once it has been proven. The representation of changing environments applies nonmonotonic reasoning, where reasoning occurs through defaults. A default follows the rule, that elements of one class

normally have a given property. These rules allow exceptions as they are not absolute. CR-Prolog is an extension of Prolog that allows describing events or exceptions that are unlikely, unusual or undesired [2]. For example epistemic disjunctions $l_1|l_2$ are possible, where one of those literals is believed to be true, as well as two types of negation. A default can be represented with $loves(P,C) \ :- \ parent(P,C),\ not\ -loves(P,C)$. Statements that contradict the default conclusion are direct exceptions to default. With CR-Prolog consistency restoring rules (cr-rules) $(l_0 \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ can be implemented, which define literals as atoms or its negation. To form an answer set, a rationality principle must be satisfied, so that the reasoner shall believe nothing, that he is not forced to believe by the program's rules. A program can obtain a consistent answer set with regular ASP-rules only, otherwise the program becomes inconsistent, because the program can't avoid contradiction with regular rules only. Then consistency restoring rules will be activated.

3 Effects of Actions, Hybrid Reasoning and Advantages of ASP

A huge challenge in ASP is representing and reasoning about the direct and indirect effects of actions. A changing domain (dynamic domain) can be described mathematically, so that effects can be predicted. To represent a discrete dynamic domain, one uses a transition diagram, where the nodes are possible physical states of the domain and the arcs are actions. Because specifying what has changed and what has not changed, also known as the frame problem, is very complex, the solution can be reduced to defining things, that normally stay as they are, which the inertia axiom says. This is achieved by defaults. Action languages are higher-level and languages for simpler representation and avoiding ASP details, that specify statement-action-statement transition diagrams. A statement of the form *A causes F if P* means that the execution of action A satisfies property P causes fluent F to become true in the resulting state. A statement of the form *F if P* means that every state satisfying property P also satisfies F. While Action languages looks different than ASP, ASP still defines the semantics of Action languages.

Hybrid reasoning is a method to integrate high-level reasoning task like planning, hypothetical reasoning or diagnosis with low-level external computations like feasibility checks of actions, extraction of relevant knowledge. This can be done by External Atoms, which are expressions of the form
g $[\mathrm{Y}_1, \ldots, Y_n](X_1, \ldots, X_m)$, where Y_i are elements of the input list and X_i are elements of the output list in the predicate external predicate g. By deciding the values of an output tuple with respect to the values of an input tuple, results of external computation can be embedded into ASP programs. Although ASP has software-engineering challenges, its readability, extensibility, ease of maintenance and flexibility are huge advantages. As complex requirements can be formalized quickly, ASP normally has a lower implementation cost and benefits in computational efficiency and optimization techniques.

4 Conclusion

ASP has many advantages in different fields. With ASP working and reasoning with commonsense knowledge becomes possible, exceptions are handled with the use of defaults and cr-rules. With the inertia axom, reasoning of effects of actions becomes less complicated and can be implemented with Action languages. In hybrid reasoning, low-level computations are integrated with external atoms and due to optimization benefits, ASP has a high computation efficiency.

References

1. Esra Erdem, Michael Gelfond, Nicola Leone: Applications of Answer Set Programming. AI Magazine 37(3): 53-68 (2016)
2. Balduccini: Cr-models: An Inference Engine for CR-Prolog https://link.springer.com/chapter/10.1007/978-3-540-72200-7_4