Declarative Rule-Based Constraint Programming

Emilia Oikarinen
Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science
P.O. Box 5400, FI-02015 HUT, Finland
Emilia.Oikarinen@hut.fi

ABSTRACT

The focus of this research is on the area of declarative rule-based constraint programming, in particular on answer set programming. Answer set programming (ASP) is an approach to rule-based constraint programming that has received increasing attention over the last few years. Rule-based programming seems like a natural way to represent specifications for several problems. The aim is to develop automated tools that enable solving hard mathematical and engineering problems declaratively. On a general level the goal is to obtain a deeper understanding of program development in ASP. In particular, the idea is to develop answer set programming into a more module-oriented direction in which ASP programs would consist of modules that are combined through suitable interfaces.

1. BACKGROUND AND MOTIVATION

Answer set programming [16, 17] is an approach to declarative rule-based constraint programming that has received increasing attention over the last few years. In answer set programming (ASP) the problem at hand is solved declaratively by writing a logic program whose answer sets correspond to the solutions of the problem, and computing the answer sets of the program using a special purpose search engine. The growing interest towards ASP is mostly due to efficient search engines such as DLV [12], SMODELS [20], and GNT [8] available today. Consequently, a variety of interesting applications of ASP has emerged. Currently ASP has applications in many areas, e.g., planning [17], product configuration [21], computer aided verification [5], wire routing in VLSI design [2] and logical cryptanalysis [6], just to mention few.

The standard syntax of logic programs is that of *normal* logic programs [15] with Clark's *negation as failure* in the bodies of rules. There are many generalizations to the basic syntax, e.g., *disjunctive rules* [7], and *weight rules* [20]. The semantics of the resulting classes of logic programs is determined by generalizations of the *stable model semantics* [3] that has settled as a leading semantics in ASP.

Despite the declarative nature of ASP, the development of programs resembles that of programs in conventional programming. That is, a programmer often develops a series of programs for a particular problem, e.g., when optimizing the execution time and space. As a consequence, there is a need to ensure that subsequent programs which differ in performance yield the same output. This gives rise to the question in what circumstances different formulations are to be considered equivalent. In [9], we develop a method for verifying the equivalence of *weight constraint programs* supported by the SMODELS system [20]. In [19, 18] we extend the same approach to a more general class of *disjunctive logic programs*. Our tools [11] enable automated verification of equivalence of these rather general classes of logic programs. There are also other notions of equivalence proposed, e.g., *strong equivalence* by Lifschitz et al. [14] and *uniform equivalence* by Eiter and Fink [1]. However, none of the current notions of equivalence has all properties desirable, so more work on the subject is still needed.

The aim of this research on a general level is to obtain a deeper understanding of program development in ASP, in particular to develop answer set programming into a more module-oriented direction in which ASP programs would consist of modules that are combined through suitable interfaces. Program optimization would thus involve optimization at the module level and therefore, for example, the concept of equivalence of logic programs should also be brought to the module level. Module-based logic programming would also enable the use of programs consisting of modules using different semantics. Such programs could be handled if suitable translations between different semantics were defined.

2. CURRENT RESEARCH

The stable model semantics of disjunctive logic programs [4] is based on classical models which are minimal with respect to subset inclusion. As a consequence, atoms in disjunctive logic programs are false by default. Often this is highly desirable, but some problems become awkward to formalize under such a blind minimization. On the other hand, *parallel circumscription* [13] allows versatile use of *minimized*, *fixed* and *varying* atoms making formalization of certain problems easier. In [10] we examine how the expressive power of parallel circumscription can be captured using disjunctive logic programming. The key result is that when the use of varying

atoms is allowed, it leads to a proper increase in expressive power.

It is typical that only certain atoms appearing in a logic program are necessary to represent the solutions of the problem formalized by the program. Other atoms act as auxiliary concepts needed in the formalization and they might not appear in other programs formulated for the problem. Consequently, the models assigned to two programs, although formulated for the same problem, may differ already on the basis of auxiliary atoms. To verify equivalence of such programs one should be able to consider only the non-auxiliary atoms in the model. This leads to the concepts of visible and invisible atoms in a logic program and to the notion of *visible equivalence*. We are currently finishing an extended journal version of [9] where the effect of visibility on the verification of equivalence is taken into account.

3. FUTURE ASPECTS

The goal is to make program development in ASP easier and in general to make ASP methodology more accessible for specialists in other fields than computer science. The near-future tasks involve considering the following questions: what properties answer set programming interfaces should have and how should they be defined, how should modularity be understood in ASP, and what properties should be expected from a notion of equivalence for modular ASP programs.

4. REFERENCES

[1] T. Eiter and M. Fink. Uniform equivalence of logic programs under the stable model semantics. In *Proc. of the 19th International Conference on Logic Programming*, pages 224–238, Mumbai, India, December 2003. Springer.

[2] E. Erdem, V. Lifschitz, and M. Wong. Wire routing and satisfiability planning. In *Proc. of the First International Conference on Computational Logic, Automated Deduction: Putting Theory into Practise*, pages 822–836, London, UK, July 2000. Springer.

[3] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. MIT Press.

[4] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[5] K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.

[6] M. Hietalahti, F. Massacci, and I. Niemelä. DES: a challenge problem for non-monotonic reasoning systems. In *Proc. of the 8th International Workshop on Non-Monotonic Reasoning*, Colorado, USA, April 2000. CoRR:cs.AI/0003039.

[7] K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.

[8] T. Janhunen et al. Unfolding partiality and disjunctions in stable model semantics. CoRR: cs.AI/0303009, March 2003.

[9] T. Janhunen and E. Oikarinen. Testing the equivalence of logic programs under stable model semantics. In *Proc. of the 8th European Conference*, pages 493–504, Cosenza, Italy, September 2002. Springer.

[10] T. Janhunen and E. Oikarinen. Capturing parallel circumscription with disjunctive logic programs. In *Proc. of the 9th European Conference*, Lissabon, Portugal, September 2004. Accepted for publification.

[11] T. Janhunen and E. Oikarinen. LPEQ and DLPEQ – translators for automated equivalence testing of logic programs. In *Proc. of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 336–340, Fort Lauderdale, USA, January 2004. Springer. System description.

[12] N. Leone et al. The DLV system for knowledge representation and reasoning. CoRR: cs.AI/0211004, September 2003.

[13] V. Lifschitz. Computing circumcription. In *Proc. of the 9th International Joint Conference on Artificial Intelligence*, pages 121–127, Los Angeles, USA, August 1985. Morgan Kaufmann.

[14] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.

[15] J. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1987.

[16] W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer, 1999.

[17] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Math. and AI*, 25(3–4):241–273, 1999.

[18] E. Oikarinen. Testing the equivalence of disjunctive logic programs. Research Report A85, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, December 2003.

[19] E. Oikarinen and T. Janhunen. Verifying the equivalence of logic programs in the disjunctive case. In *Proc. of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 180–193, Fort Lauderdale, USA, January 2004. Springer.

[20] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[21] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proc. of the First International Workshop on Practical Aspects of Declarative Languages*, pages 305–319, Texas, USA, January 1999. Springer.