Logic Programs with Preferences

Marcello Balducini

October 18^{th}, 2000

1 Introduction

In this seminar, we compared two approaches to the introduction of preferences among rules in logic programs ([1], [2]).

2 Summary of the Papers

In [1], Schaub et at. present an encoding of preferences among rules in logic programs based on the introduction of *preference atoms* of the form $n_1 \prec n_2$. Terms n_1 and n_2 identify rules of the program. The informal meaning of the atom is that, *in case both rules can be applied when computing an answer set, the rule denoted by n_2 has to be applied first.* The semantics of ordered logic programs is given in terms of the semantics of regular logic programs, by means of a syntactical transformation $\mathcal{T}$ that maps ordered logic programs into regular ones.

The approach described in [2] introduces preferences on defaults only. Standard A-Prolog atoms are used to encode the preference between rules and the semantics of preference atoms is given in terms of a set of A-Prolog rules. The informal semantics of a fact describing that default d_2 is preferred over d_1 is that, *in case both can be applied, only d_2 must be applied.* These axioms defining the formal semantics of preference atoms are added to the ordered logic program when answer sets are to be computed.

3 Comparison

From the point of view of syntax, the two approaches are very similar. Both deal with preference atoms that can appear in the head as well as in the body of rules. Rules whose head is a preference atoms are allowed to have a non-empty body. The name of rules is, in both cases, a term, rather than a constant symbol. It is easy to see that, if the approach presented in [1] is restricted to defaults, the two ways of expressing preferences are syntactically equivalent.

From the point of view of semantics, [1] and [2] present two alternative views of logic programs with preferences. The view in [1] is more procedural. Preferences impose constraints on the order in which rules are applied during the computation of answer sets. On the contrary, the semantics described in [2] only affects whether defaults are applied or not. There is no relation to how answer sets are computed, which makes the approach more declarative.

A further difference is that, in [1], preferences can be expressed with respect to all rules, while, in [2], they can be expressed on defaults only. The former choice gives apparently unintuitive results in some cases in which preferences over non-default rules are involved. For example, under the first semantics, the following program Π:

$$\begin{array}{llll} & p. & & \\ \mathbf{r}_1: & f & :- & p. \\ \mathbf{r}_2: & \neg f & :- & p. \\ & r_2 \prec r_1. & & \end{array}$$

has no answer set, while, intuitively, preference rule $r_2 \prec r_1$ should let us obtain one (unique) answer set $\{p, \neg f\}$. The reason is that preferring rule r_2 over rule r_1 only means that rule r_2 must be applied *before* r_1, in case both can be applied. In the previous example, after applying r_2, r_1 can still be applied, which causes contradiction. On the other hand, suppose to be to apply the semantics of [2] also to non-default rules (the extension is straightforward). Then, under this semantics, Π would have only one answer set, $\{p, \neg f\}$.

From the perspective of computational efficiency, the transformation described in [1] has major problems when applied to programs containing several preferences. This is due in particular to the structure of rule $c_1(r)$, generated during transformation $\mathcal{T}$. Its body contains all preference atoms involving rule r. In the case of names of rules containing variables, the body of $c_1(r)$ clearly becomes huge.

Finally, it has to be noted that one of the concluding remarks in [1] is not correct. In brief, the authors say that the use of Prolog-like list notation for preference atoms, in [2], is problematic for the use in systems like SMODELS and DLV. This is not indeed the case. In fact, atoms of the form

$$default(n, p, [r, \neg s], [q]).$$

can be immediately rewritten as:

$$\begin{array}{l} default(n). \\ head(n, p). \\ p\_prec(n, r). \\ p\_prec(n, \neg s). \\ n\_prec(n, q). \end{array}$$

which presents no problems for SMODELS and DLV.

References

[1] James P. Delgrande, Torsten Schaub, and Hans Tompits. Logic programs with compiled preferences. In *Proceedings of the European Conference on Artificial Intelligence*, pages 392–398. IOS Press, 2000.

[2] Michael Gelfond and Son Cao Tran. Reasoning with prioritized defaults. In *Third International Workshop, LPKR'97*, volume 1471 of *Lecture Notes in Artificial Intelligence (LNCS)*, pages 164–224, Oct 1997.