Order and Negation as Failure

Davy Van Nieuwenborgh* and Dirk Vermeir**

Dept. of Computer Science
Vrije Universiteit Brussel, VUB
{dvnieuwe,dvermeir}@vub.ac.be

Abstract. We equip ordered logic programs with negation as failure, using a simple generalization of the preferred answer set semantics for ordered programs. This extension supports a convenient formulation of certain problems, which is illustrated by means of an intuitive simulation of logic programming with ordered disjunction. The simulation also supports a broader application of “ordered disjunction”, handling problems that would be cumbersome to express using ordered disjunction logic programs. Interestingly, allowing negation as failure in ordered logic programs does not yield any extra computational power: the combination of negation as failure and order can be simulated using order (and true negation) alone.

1 Introduction

Non-monotonic reasoning using logic programming can be accomplished using one of several possible extensions of positive programs. The better known extension is negation as failure which has a long history, starting from the Clark completion [7], over stable model semantics [11] and well-founded semantics [25], to answer set programming [20]. It is well-known that adding negation as failure to programs results in a more expressive formalism. However, in the context of disjunctive logic programming [21, 19, 14] demonstrated that adding negation as failure positively in a program, i.e. in the head of the rules, yields no extra computational power to the formalism. One of the more interesting features of negation as failure in the head is that answers no longer have to be minimal w.r.t. subset inclusion (e.g. the program \{a \lor not a \leftarrow\} has both \{a\} and \emptyset as answer sets). Indeed, such minimality turns out to be too demanding to express certain problems, e.g. in the areas of abductive logic programming [15,13] or logic programming with ordered disjunction [14].

Introducing (preference) order in logic programs represents another way to naturally express many "non-monotonic" problems. Many proposals [17,18,24,33,5,28,8,27,1] for logic programming extensions incorporate some kind of order, sometimes in a rather subtle way.

The preferred answer set semantics defined in [27], uses a partial order defined among the rules of a simple program, i.e. a non-disjunctive program containing only

* Supported by the FWO.
** This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project.

C. Palamidessi (Ed.): ICLP 2003, LNCS 2916, pp. 194–208, 2003.
© Springer-Verlag Berlin Heidelberg 2003
classical negation, to prefer certain extended answer sets of the program above others, where the extended answer sets semantics naturally extends the classical one \cite{20} to deal with inconsistencies in a program by allowing contradictory rules to defeat each other. It turns out that such an order can simulate negation as failure in both seminegative logic programs, where only rule bodies may contain negation as failure, under the stable model semantics \cite{11}, and disjunctive logic programs under the possible model semantics \cite{23}, demonstrating that order is at least as expressive as negation as failure.

Often, the introduction of order increases the expressiveness of a logic programming formalism, as illustrated by the complexity results in e.g. \cite{16,6,3,27}. A natural question to ask then is whether combining order and negation as failure yields even more expressiveness. For the case of the ordered programs from \cite{27}, this paper will show that the answer to the above question is negative.

In this paper, we first extend the preferred answer set semantics for ordered programs to extended ordered programs, i.e. programs containing both classical negation and negation as failure combined with an order relation among the rules. Just as for disjunctive logic programming, adding negation as failure positively results in a formalism where answer sets are not anymore guaranteed to be subset minimal. Then we show, by means of a transformation, that the preferred answer set semantics for such extended ordered programs can be captured by the one for ordered programs (without negation as failure).

Despite the fact that extended ordered programs do not yield any extra computational power, they can be used profitably to express certain problems in a more intuitive and natural way. We demonstrate this by providing an elegant transformation from logic programs with ordered disjunction into extended ordered programs.

Logic programming with ordered disjunction \cite{1} is a combination of qualitative choice logic \cite{2} and answer set programming \cite{20}. Instead of an order relation on the rules in a program, this formalism uses an order among the head literals of disjunctive rules. Intuitively, this relation, called ordered disjunction, ranks the conclusions in the head of rules. A lesser alternative should only be chosen if all higher ranked options could not be fulfilled.

Preferred answer sets for programs with ordered disjunction need not be subset minimal. E.g. for the program $(\times$ is used for disjunction and more preferred alternatives come first in the head of a rule)
\[
\begin{align*}
a \times b & \leftarrow \\
c \times b & \leftarrow a \\
\neg c & \leftarrow
\end{align*}
\]
both $S = \{b, \neg c\}$ and $T = \{a, b, \neg c\}$ are preferred answer sets, although $S \subset T$. Hence, a translation to a class of programs with subset minimal (preferred) answer sets is impossible unless e.g. extra atoms are introduced.

In \cite{4} a procedure to compute the preferred answer sets of a logic program with ordered disjunction is presented. The algorithm uses two different programs: one to generate candidate answer sets and another one to test whether those candidates are preferred w.r.t. the order on the literals in the ordered disjunctions. Using our transformation into extended ordered logic programs, combined with the further translation that removes negation as failure, the algorithm to compute answer sets for ordered programs \cite{27} can be used to do the same for logic programs with ordered disjunction.