Implementing Ordered Disjunction
Using Answer Set Solvers for Normal Programs

Gerhard Brewka¹, Ilkka Niemelä², and Tommi Syrjänen²

¹ Universität Leipzig, Institut für Informatik
Augustusplatz 10-11, 04109 Leipzig, Germany
brewka@informatik.uni-leipzig.de
² Helsinki University of Technology, Dept. of Computer Science and Engineering
Lab. for Theoretical Computer Science
P.O.Box 5400 FIN-02015 HUT, Finland
{Ilkka.Niemela,Tommi.Syrjanen}@hut.fi

Abstract. Logic programs with ordered disjunction (LPODs) add a new
connective to logic programming. This connective allows us to repre-
sent alternative, ranked options for problem solutions in the heads of
rules: $A \times B$ intuitively means: if possible $A$, but if $A$ is not possible, then
at least $B$. The semantics of logic programs with ordered disjunction is
based on a preference relation on answer sets. In this paper we show
how LPODs can be implemented using answer set solvers for normal
programs. The implementation is based on a generator which produces
candidate answer sets and a tester which checks whether a given candi-
date is maximally preferred and produces a better candidate if it is not.
We also discuss the complexity of reasoning tasks based on LPODs.

1 Introduction

In [1] a propositional logic called Qualitative Choice Logic (QCL) is introduced.
The logic contains a new connective $\times$ representing ordered disjunction. Intu-
itably, $A \times B$ stands for: if possible $A$, but if $A$ is impossible then (at least) $B$. In
[2] it is shown how ordered disjunction can be added to logic programs with two
kinds of negation under answer set semantics. The resulting logic programs with
ordered disjunction (LPODs for short) allow us to combine default knowledge
with knowledge about preferences in a simple and elegant way.

In this paper we show how LPODs can be implemented using answer set
solvers (ASP solvers) for normal (non-disjunctive) programs. This means that
when implementing LPODs it is possible to directly exploit constantly improving
performance of ASP solvers for standard logic programs such as Smodels and
dlv. The implementation is based on two normal logic programs, a generator
which produces candidate answer sets and a tester which checks whether a given
candidate is maximally preferred. The tester produces a better answer set if the
candidate is not preferred. Iteration thus leads to a maximally preferred answer
set. We also discuss the complexity of reasoning tasks based on LPODs.

We will restrict our discussion in this paper to propositional programs. How-
ever, as usual in answer set programming, we admit rule schemata containing
variables bearing in mind that these schemata are just convenient representations for the set of their ground instances.

We have constructed a prototype implementation for LPODs based on Smodels, an efficient ASP solver developed at Helsinki University of Technology. The generator and tester programs use special rule types of the Smodels system, but they can be modified to work with any ASP solver. The prototype implementation is available at http://www.tcs.hut.fi/Software/smodels/priority.

The rest of the paper is organized as follows. In the next section we recall the basic notions underlying syntax and semantics of LPODs. For a more detailed discussion the reader is referred to [2]. Section 3 discusses several alternative preference relations on answer sets which can be obtained based on the satisfaction degrees of rules. Section 4 presents our Smodels based implementation. Section 5 gives complexity results. Section 6 gives a short discussion on applying preferences on the problem of configuration management. Section 7 concludes.

2 Logic Programs with Ordered Disjunction

Logic programming with ordered disjunction is an extension of logic programming with two kinds of negation (default and strong negation) [4]. The new connective $\times$ representing ordered disjunction is allowed to appear in the head of rules only. A (propositional) LPOD thus consists of rules of the form

$$C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k$$

where the $C_i$, $A_j$, and $B_l$ are ground literals.

The intuitive reading of the rule head is: if possible $C_1$, if $C_1$ is not possible, then $C_2$, ..., if all of $C_1, \ldots, C_{n-1}$ are not possible, then $C_n$. The literals $C_i$ are called choices of the rule. Extended logic programs are a special case where $n = 1$ for all rules. We omit $\leftarrow$ whenever $m = 0$ and $k = 0$. Moreover, rules of the form $\leftarrow$ body (constraints) are used as abbreviations for $p \leftarrow \neg p$, body for some $p$ not appearing in the rest of the program. The effect is that no answer sets containing body exist. We use the notations $\text{At}(P)$ and $\text{Lit}(P)$ to denote the sets of atoms and literals occurring in a LPOD $P$.

As discussed in [2] answer sets of LPODs cannot be inclusion minimal because this would in certain cases exclude answer sets from consideration which, intuitively, satisfy the rules best. The definition of answer sets for LPODs is therefore based on the notion of a split program. This notion was first used in [7] for disjunctive logic programs. A split program consists of single head rules obtained from the original program by picking one of the available alternatives. Our definition of split programs for LPODs differs in two respects from Sakama and Inoue’s to comply with the intuitive reading of ordered disjunction:

1. we require that a split program contains exactly one of the alternatives provided in the original program by a single rule,
2. our single head rules are slightly more complicated to guarantee that a choice is only made if a better choice isn’t already derived through some other rule.