Implementing Ordered Disjunction Using Answer Set Solvers for Normal Programs

Gerhard Brewka[1], Ilkka Niemelä[2], and Tommi Syrjänen[2]

[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 (*LPOD*s) add a new connective to logic programming. This connective allows us to represent 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 *LPOD*s 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 candidate is maximally preferred and produces a better candidate if it is not. We also discuss the complexity of reasoning tasks based on *LPOD*s.

1 Introduction

In [2] a propositional logic called Qualitative Choice Logic (*QCL*) is introduced. The logic contains a new connective $\times$ representing ordered disjunction. Intuitively, $A \times B$ stands for: if possible A, but if A is impossible then (at least) B. In [1] 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 (*LPOD*s for short) allow us to combine default knowledge with knowledge about preferences in a simple and elegant way.

In this paper we show how *LPOD*s can be implemented using answer set solvers (ASP solvers) for normal (non-disjunctive) programs. This means that when implementing *LPOD*s 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 *LPOD*s.

We will restrict our discussion in this paper to propositional programs. However, 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 *LPOD*s 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 *LPOD*s. For a more detailed discussion the reader is referred to [1]. 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

$$C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \text{not } B_1, \ldots, \text{not } B_k$$

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 $\leftarrow \mathit{body}$ (constraints) are used as abbreviations for $p \leftarrow \text{not } p, \mathit{body}$ 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 $\mathit{At}(P)$ and $\mathit{Lit}(P)$ to denote the sets of atoms and literals occurring in a *LPOD* P.

As discussed in [1] answer sets of *LPOD*s 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 *LPOD*s 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 *LPOD*s 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.

The precise definition is as follows:

Definition 1. *Let $r = C_1 \times \cdots \times C_n \leftarrow body$ be a rule. For $k \leq n$ we define the kth option of r as*

$$r^k = C_k \leftarrow body, \text{not } C_1, \ldots, \text{not } C_{k-1}.$$

Definition 2. *Let P be an LPOD. Then P' is a split program of P if it is obtained from P by replacing each rule in P by one of its options.*

Here is a simple example. Let P consist of the rules

$$\begin{aligned} A \times B &\leftarrow \text{not } C \\ B \times C &\leftarrow \text{not } D \end{aligned} \tag{1}$$

We obtain 4 split programs

$$\begin{array}{ll} A \leftarrow \text{not } C & A \leftarrow \text{not } C \\ B \leftarrow \text{not } D & C \leftarrow \text{not } D, \text{not } B \\ & \\ B \leftarrow \text{not } C, \text{not } A & B \leftarrow \text{not } C, \text{not } A \\ B \leftarrow \text{not } D & C \leftarrow \text{not } D, \text{not } B \end{array}$$

Split programs do not contain ordered disjunction. We thus can define:

Definition 3. *Let P be an LPOD. A set of literals A is an answer set of P if it is a consistent answer set of a split program P' of P.*

We exclude inconsistent answer sets from consideration since they do not represent possible problem solutions. In the example above we obtain 3 answer sets: $\{A, B\}, \{C\}, \{B\}$. Note that one of the answer sets is a proper subset of another answer set. On the other hand, none of the rules in the original *LPOD* sanctions more than one literal in any of the answer sets, as intended.

Not all of the answer sets satisfy our most intended options. Clearly, $\{A, B\}$ gives us the best options for both rules, whereas $\{C\}$ gives only the second best option for the latter and $\{B\}$ the second best option for the former rule. We therefore introduce the notion of a preferred answer set in the next section.

3 Preferred Answer Sets

To distinguish between more and less intended answer sets we introduce the degree of satisfaction of a rule in an answer set:

Definition 4. *Let S be an answer set of an LPOD P. Then S satisfies the rule*

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m, \text{not } B_1, \ldots, \text{not } B_k$$

- *to degree 1 if $A_j \notin S$, for some j, or $B_i \in S$, for some i,*
- *to degree j $(1 \leq j \leq n)$ if all $A_j \in S$, no $B_i \in S$, and $j = min\{r \mid C_r \in S\}$.*

The degrees can be viewed as penalties: the higher the degree the less satisfied we are. If the body of a rule is not satisfied, then there is no reason to be dissatisfied and the best degree 1 is obtained. We denote the degree of r in S by $deg_S(r)$.

The satisfaction degrees of the rules of a program P are the basis for defining a preference relation on the answer sets of P. There are many different ways of inducing such a preference relation. We will discuss three of them in this paper.

The first preference criterion is based on the cardinality of the sets of rules satisfied to a particular degree. For a set of literals S, let $S^i(P) = \{r \in P \mid deg_S(r) = i\}$. Now cardinality based preference can be defined as follows:

Definition 5. *Let S_1 and S_2 be answer sets of an LPOD P. Then S_1 is cardinality-preferred to S_2 ($S_1 >_c S_2$) iff there is i such that $|S_1^i(P)| > |S_2^i(P)|$, and for all $j < i$, $|S_1^j(P)| = |S_2^j(P)|$.*

In certain applications counting does not provide the best way of defining preferences among answer sets. We therefore propose a second, inclusion based criterion. This is the criterion originally used in [1]:

Definition 6. *Let S_1 and S_2 be answer sets of an LPOD P. The answer set S_1 is inclusion-preferred to S_2 ($S_1 >_i S_2$) iff there is k such that $S_2^k(P) \subset S_1^k(P)$, and for all $j < k$, $S_1^j(P) = S_2^j(P)$.*

Although inclusion-preference is more cautious than cardinality-preference it is sometimes not cautious enough; in some cases adding unattainable preferences changes the set of preferred answer sets. Consider the following decision over possible desserts:

$$\begin{aligned} r_1 &: \textit{ice-cream} \times \textit{cake} \\ r_2 &: \textit{coffee} \times \textit{tea} \\ r_3 &: \ \leftarrow \textit{coffee}, \textit{ice-cream} \end{aligned} \tag{2}$$

Now there are two preferred answer sets, $\{\textit{ice-cream}, \textit{tea}\}$ and $\{\textit{coffee}, \textit{cake}\}$. Neither of them dominates the other because they both satisfy one rule to the first degree and one rule to the second degree. Now replace r_1 by

$$r_1' : \textit{cookie} \times \textit{ice-cream} \times \textit{cake}$$

and add the fact $\neg\textit{cookie}$ to the program. Then, $\{\textit{coffee}, \textit{cake}, \neg\textit{cookie}\}$ is the only preferred answer set. By adding an unsatisfiable preference to cookies, we inadvertently made the second preference more important than the first. To avoid effects of this kind one can use the following Pareto criterion[1]:

Definition 7. *Let S_1 and S_2 be answer sets of an LPOD P. Then S_1 is Pareto-preferred to S_2 ($S_1 >_p S_2$) iff there is $r \in P$ such that $deg_{S_1}(r) < deg_{S_2}(r)$, and for no $r' \in P$ $deg_{S_1}(r') > deg_{S_2}(r')$.*

The proof of the following proposition is straightforward:

[1] The Pareto-criterion was suggested to us by Harri Haanpää.

Proposition 1. *Let S_1 and S_2 be answer sets of an LPOD P. Then $S_1 >_p S_2$ implies $S_1 >_i S_2$ and $S_1 >_i S_2$ implies $S_1 >_c S_2$.*

Definition 8. *A set of literals S is a k-preferred (where $k \in \{c, i, p\}$) answer set of an LPOD P iff S is an answer set of P and there is no answer set S' of P such that $S' >_k S$.*

Given a particular preference criterion k, we say a literal l is a conclusion of an *LPOD* P iff l is contained in all k-preferred answer sets of P. In many applications, for instance in design and configuration, each preferred answer set represents a solution that satisfies the preferences best.

Sometimes we may want to express that one preference is more important than another. Consider again the dessert program (2) in its original form. Assume that we would rather have ice-cream than coffee. Now we would like have a mechanism that could express the differences in preference importance. A convenient way is to express these meta-preferences by defining a relation $\succ$ on rules. In our case we could simply say $r_1 \succ r_2$. It is not difficult to take these rule preferences into account. Let us illustrate this using Pareto preference:

Definition 9. *Let S_1 and S_2 be answer sets of an LPOD P, $\succ$ a preference relation on the rules of P. S_1 is Pareto-preferred to S_2 ($S_1 >_p S_2$) iff*

1. *there is $r \in P$ such that $deg_{S_1}(r) < deg_{S_2}(r)$, and*
2. *for each $r' \in P$: $deg_{S_1}(r') > deg_{S_2}(r')$ implies there is r'' such that $r'' \succ r'$ and $deg_{S_1}(r'') < deg_{S_2}(r')$.*

In principle, it is possible to represent preferences among rules within the programs and thus to make them context dependent. However, since the rule preference information then is part of the generated answer sets one has to be careful not to loose anti-symmetry. For instance, if we have

$$\begin{array}{ll} r_1 : a \times \neg a & r_2 \succ r_1 \leftarrow a \\ r_2 : \neg a \times a & r_1 \succ r_2 \leftarrow \neg a \end{array}$$

then, from the perspective of each of the two answer sets, the other answer set is preferred. As long as different answer sets do not disagree about preferences among rules, one is on the safe side. We will not pursue this issue further here.

4 Implementation

We can compute preferred answer sets of an *LPOD* P using standard answer set implementations and two programs. A similar approach is used in [5] to compute stable models of disjunctive programs using *Smodels*. The two programs are:

– A *generator* $G(P)$ whose stable models correspond to the answer sets of P; and
– A *tester* $T(P, M)$ for checking whether a given answer set M of P is preferred.

The two programs are run in an interleaved fashion. First, the generator constructs an answer set M of P. Next, the tester tries to find an answer set M' that is strictly better than M. If there is no such M', we know that M is a preferred answer set. Otherwise, we use $G(P)$ to construct the next candidate. When we want to find only one preferred answer set we can save some effort by taking M' directly as the new answer set candidate.

The basic idea of $G(P)$ is to encode all possible split programs of P into one program by adding an explicit choice over the options of each ordered disjunction. We encode the choice using new atoms of the form $c(r,k)$ to denote that we are using the kth option of rule r.

To make model comparison easier we also add another set of new atoms, $s(r,k)$ to denote that the rule r is satisfied to the degree k. These atoms are not strictly necessary but they make the programs more readable.

Definition 10. *Let P be an LPOD and $r = C_1 \times \cdots \times C_n \leftarrow body$ be a rule in P. Then the translation $G(r,k)$ of the kth option of r is defined as follows:*

$$\begin{aligned} G(r,k) = \{ & C_k \leftarrow c(r,k), \text{not } C_1, \ldots, \text{not } C_{k-1}, body; && (3)\\ & \leftarrow C_k, \text{not } c(r,k), \text{not } C_1, \ldots, \text{not } C_{k-1}, body\} && (4)\end{aligned}$$

The satisfaction translation $S(r)$ is:

$$\begin{aligned} S(r) = & \{s(r,1) \leftarrow \text{not } c(r,1), \ldots, \text{not } c(r,n)\} \cup && (5)\\ & \{\, s(r,i) \leftarrow c(r,i) \mid 1 \leq i \leq n\} && (6)\end{aligned}$$

The translation $G(r)$ is:

$$G(r) = \big\{1\ \{c(r,1), \ldots, c(r,n)\}\ 1 \leftarrow body\big\} \cup \bigcup\{G(r,k) \mid k \leq n\} \cup S(r) \qquad (7)$$

The generator $G(P)$ is defined as follows:

$$G(P) = \bigcup\{G(r) \mid r \in P\} \qquad (8)$$

The definition of $G(P)$ is rather straightforward, but a few points may need explaining. First, the rule $1\ \{c(r,1), \ldots, c(r,n)\}\ 1 \leftarrow body$ states that if *body* is true, then exactly one of the atoms $c(r,k)$ is true. Its formal semantics is defined in [6] but it can also be seen as a shorthand for n pairs of rules of the form:

$$\begin{aligned} c(r,k) &\leftarrow \text{not } \neg c(r,k), body\\ \neg c(r,k) &\leftarrow \text{not } c(r,k)\end{aligned}$$

and $n^2 - n$ constraints $\leftarrow c(r,i), c(r,j), i \neq j$.

Also, the reason for having two rules in $G(r,k)$ may not be clear, since (3) already ensures that only correct answer sets of the split program P' will be generated. Now consider the situation where some C_j, $j < k$, is a consequence of a different part of the program. Then without (4) we could have an answer set where $c(r,k)$ is true, but C_k is not because C_j blocks (3). In other words, we would have chosen to satisfy r to the degree k, but it actually would be satisfied to the degree j. The rule (4) prevents this unintuitive behavior by always forcing us to choose the lowest possible degree.

Example 1. The program (1) is translated to:

$$
\begin{array}{rlrl}
1\ \{c(1,1), c(1,2)\}\ 1 & \leftarrow \text{not } C & A & \leftarrow c(1,1), \text{not } C \\
1\ \{c(2,1), c(2,2)\}\ 1 & \leftarrow \text{not } D & B & \leftarrow c(2,1), \text{not } D \\
 & \leftarrow \text{not } c(1,1), A, \text{not } C & B & \leftarrow c(1,2), \text{not } A, \text{not } C \\
 & \leftarrow \text{not } c(1,2), B, \text{not } A, \text{not } C & C & \leftarrow c(2,2), \text{not } B, \text{not } D \\
 & \leftarrow \text{not } c(2,1), B, \text{not } D & s(1,1) & \leftarrow \text{not } c(1,1), \text{not } c(1,2) \\
 & \leftarrow \text{not } c(2,2), C, \text{not } B, \text{not } D & s(2,1) & \leftarrow \text{not } c(2,1), \text{not } c(2,2) \\
s(1,1) \leftarrow c(1,1) & \quad s(1,2) \leftarrow c(1,2) & s(2,1) & \leftarrow c(2,1) \quad s(2,2) \leftarrow c(2,2)
\end{array}
$$

It has three answer sets: $\{A, B, c(1,1), c(2,1), s(1,1), s(2,1)\}$, $\{B, c(1,2), c(2,1), s(1,2), s(2,1)\}$, and $\{C, c(2,2), s(1,1), s(2,2)\}$. Note that in the last case there is no atom $c(1,k)$ since the body of the first rule is not satisfied.

Proposition 2. *Let P be an LPOD. Then M is an answer set of $G(P)$ if and only if $M \cap Lit(P)$ is an answer set of P.*

Proof. Let M be an answer set of P. Now, for each rule $r = C_1 \times \cdots \times C_n \leftarrow$ *body* define $p(M,r) = \{c(r,k), s(r,k) \mid \textit{body} \text{ is satisfied in } M, C_k \in M, \text{ and } \forall i < k : C_i \notin M\} \cup \{s(r,1) \mid r \in P \text{ and } \textit{body} \text{ is unsatisfied in } M\}$. Let $M' = M \cup \bigcup_{r \in P} p(M,r)$. By definition of $p(M,r)$, M' satisfies all rules (3)–(6). Finally, (7) is satisfied since exactly one atom $c(r,k)$ was added to M' for each rule r that had its body true. It follows that M' is an answer set of $G(P)$. Now, suppose that M' is an answer set of $G(P)$. Then for each atom $C \in M = M' \cap Lit(P)$, there exists a rule of the form (3) where C is the head and some atom $c(r,k) \in M'$ occurs positively in the body. We define $P' = \{r^k \mid c(r,k) \in M'\} \cup \{r^1 \mid \neg\exists k : c(r,k) \in M'\}$. We see that P' is a split program of P that generates M as its answer set so M is an answer set of P.

Since we have three different optimality criteria, we need three different tester programs. They all consist of a common core $C(P,M)$ augmented by case-specific rules T_c, T_i, or T_p. Since we want the tester to find an answer set M' that is strictly better than a given M, we define two new atoms, namely *better* and *worse* with the intuition that *better* (*worse*) is true when M' is in some aspect better (worse) than M. If both are true, then the answer sets are incomparable.

Definition 11. *Let P be an LPOD. Then the core tester $C(P,M)$ is defined as follows:*

$$
\begin{aligned}
C(P,M) = G(P) &\cup \{o(r,k) \mid s(r,k) \in M\} \cup \{\textit{rule}(r) \leftarrow \mid r \in P\} \\
&\cup \{\textit{degree}(d) \leftarrow \mid \exists r \in P \textit{such that } r \textit{ has at least } d \textit{ options}\} \\
&\cup \{\leftarrow \text{not } \textit{better};\ \leftarrow \textit{worse}\}
\end{aligned}
$$

The k-preference tester ($k \in \{c, i, p\}$) $T_k(P,M)$ is defined as follows:

$$T_k(P,M) = C(P,M) \cup T_k$$

$$\begin{aligned}
\mathit{better} &\leftarrow s(R,I), o(R,J), I < J, \mathit{rule}(R), \mathit{degree}(I), \mathit{degree}(J)\\
\mathit{worse} &\leftarrow s(R,J), o(R,I), I < J, \mathit{rule}(R), \mathit{degree}(I), \mathit{degree}(J)
\end{aligned}$$

Fig. 1. The Pareto-preference tester T_p

$$\begin{aligned}
\mathit{better}(D_1) &\leftarrow s(R,D_1), o(R,D_2), D_1 < D_2, \mathit{rule}(R), \mathit{degree}(D_1), \mathit{degree}(D_2)\\
\mathit{worse}(D_1) &\leftarrow s(R,D_2), o(R,D_1), D_1 < D_2, \mathit{rule}(R), \mathit{degree}(D_1), \mathit{degree}(D_2)\\
\mathit{better} &\leftarrow \{\mathit{worse}(D_2) : \mathit{degree}(D_2) \wedge D_2 \leq D_1\}\ 0, \mathit{better}(D_1), \mathit{degree}(D_1)\\
\mathit{worse} &\leftarrow \{\mathit{better}(D_2) : \mathit{degree}(D_2) \wedge D_2 \leq D_1\}\ 0, \mathit{worse}(D_1), \mathit{degree}(D_1)
\end{aligned}$$

Fig. 2. The inclusion preference tester T_i

The case-specific parts of the three different testers are shown in Figures 1–3. The atoms $o(r,k)$ are used to store the degrees of satisfaction in the original answer set M so that $o(r,k)$ is added as a fact to $T(P,M)$ whenever $s(r,k) \in M$.

The p-preference tester T_p (Fig. 1) is by far the simplest. It states that M' is better if there exists some rule that has a lower degree of satisfaction in M' than in M, and worse if some rule has a higher degree. The i-preference tester T_i (Fig. 2) considers each different degree of satisfaction separately. Now M' is preferred over M if there exists some degree k that M' satisfies better and for all degrees $k' \leq k$ M' is not worse than M. The c-preference tester T_c (Fig. 3) is quite similar to T_i but we add new atoms $\mathit{s\text{-}card}(k,n)$ and $\mathit{o\text{-}card}(k,n)$ to encode the cardinalities of the sets S^k, and make the comparisons based on them.

Proposition 3. *Let P be an LPOD and M be an answer set of $G(P)$. Then M' is an answer set of $T_k(P,M)$ ($k \in \{c,i,p\}$) iff $M' \cap \mathit{Lit}(P)$ is an answer set of P which is k-preferred to $M \cap \mathit{Lit}(P)$.*

Proof. (For p-preference) First, suppose that $T_p(P,M)$ has an answer set M'. Then $\mathit{better} \in M'$ and $\mathit{worse} \notin M'$. We see that better is true exactly when $\exists r : \mathit{deg}_{M'}(r) < \mathit{deg}_M(r)$. Since $\mathit{worse} \notin M'$, we know that $\neg\exists r : \mathit{deg}_{M'}(r) >$

$$\begin{aligned}
\mathit{s\text{-}card}(K,N) &\leftarrow N\ \{s(R,N) : \mathit{rule}(R)\}\ N, \mathit{degree}(K)\\
\mathit{o\text{-}card}(K,N) &\leftarrow N\ \{o(R,N) : \mathit{rule}(R)\}\ N, \mathit{degree}(K)\\
\mathit{better}(D) &\leftarrow \mathit{s\text{-}card}(D,N_1), \mathit{o\text{-}card}(D,N_2), N_1 > N_2, \mathit{degree}(D)\\
\mathit{worse}(D) &\leftarrow \mathit{s\text{-}card}(D,N_1), \mathit{o\text{-}card}(D,N_2), N_1 < N_2, \mathit{degree}(D)\\
\mathit{better} &\leftarrow \mathit{better}(D), \mathit{degree}(D)\\
\mathit{worse} &\leftarrow \{\mathit{better}(D_2) : \mathit{degree}(D_2) \wedge D_2 < D_1\}\ 0, \mathit{worse}(D_1), \mathit{degree}(D_1)
\end{aligned}$$

Fig. 3. The cardinality preference tester T_c

$$
\begin{aligned}
\mathit{better}(R) &\leftarrow s(R,I), o(R,J), I < J, \mathit{rule}(R), \mathit{degree}(I), \mathit{degree}(J)\\
\mathit{worse} &\leftarrow s(R,J), o(R,I), \text{not } \mathit{excused}(R), I < J, \mathit{rule}(R), \mathit{degree}(I), \mathit{degree}(J)\\
\mathit{excused}(R_1) &\leftarrow R_2 \succ R_1, \mathit{better}(R_2), \mathit{rule}(R_1), \mathit{rule}(R_2)\\
\mathit{better} &\leftarrow \mathit{better}(R), \mathit{rule}(R)
\end{aligned}
$$

Fig. 4. A meta-preference tester

$deg_M(r)$ so $M' \cap Lit(P) >_p M$. Conversely, if there exists $M' >_p M$, then M' is generated by the $G(P)$ part of $T(P, M)$ so $M' \cup \bigcup_{r \in P} p(M', r) \cup \{\mathit{better}\}$ is an answer set of $T_p(P, M)$. The cases for c- and i-preference are analogous.

The following corollary is immediate from Proposition 3:

Corollary 1. *Let P be an LPOD and M be an answer set of $G(P)$. Then M is k-preferred ($k \in \{c, i, p\}$) iff $T_k(P, M)$ does not have any answer set.*

We can handle meta-preferences by modifying the rules of the *worse* atom. We define a new predicate $\mathit{excused}(r)$ to denote that some more important rule r', $r' \succ r$, is satisfied to a lower degree in M' than it was in M so we allow r to be satisfied to a higher degree. Figure 4 shows how the p-preference tester has to be modified to take the meta-information into account. The i-preference tester can be altered in a similar fashion. However, adding meta-preferences to the c-preference tester is more complex and we do not discuss it here.

5 Complexity

In the previous section we defined two extended logic programs that can be interleaved to compute k-preferred answer sets. Would it be possible to compute them using a single disjunction-free program? Unfortunately, this is impossible in the general case unless the polynomial hierarchy collapses; credulous reasoning on *LPOD*s is $\mathbf{\Sigma_2^P}$-complete for $\{i, p\}$-preferences. The c-preference is slightly easier computationally and it stays in $\mathbf{\Delta_2^P}$. Whether it is $\mathbf{\Delta_2^P}$-complete or not is still an open question. In this section we prove these complexity results.

We start by noting that since the three different $>_k$ relations are all anti-symmetric and well-founded, an *LPOD* P has at least one k-preferred answer set if it has any answer sets at all.

Theorem 1. *Let P be an LPOD. Then deciding whether P has a k-preferred ($k \in \{c, i, p\}$) answer set is* **NP**-*complete.*

Theorem 2. *Let P be an LPOD and M an answer set of P. Then deciding whether M is k-preferred ($k \in \{c, i, p\}$) is* **coNP**-*complete.*

Proof. Inclusion: If M is not a k-preferred answer set of P, then there exists an answer set M' such that $M' >_k M$. Since we can find M' with one query to an NP-oracle, showing that M is preferred is in **coNP**.

Hardness: Given a 3-SAT-instance S, we can construct an *LPOD* P with the property that $M = \{\neg sat, d\}$ is a k-preferred answer set exactly when S is unsatisfiable. This translation $t(S)$ is as follows:

$$\begin{aligned} t(S) =& \{sat \times \neg sat; d \leftarrow \text{not } \neg d; \neg d \leftarrow \text{not } d; \neg sat \leftarrow d; sat \leftarrow \text{not } \neg sat\} \\ & \cup \{\neg sat \leftarrow \text{not } A_1, \text{not } A_2, \text{not } A_3 \mid A_1 \vee A_2 \vee A_3 \in S\} \\ & \cup \{a \leftarrow \text{not } \neg a, \neg d; \neg a \leftarrow \text{not } a, \neg d \mid a \in At(S)\} \end{aligned}$$

Theorem 3. *Given an LPOD P and a literal $l \in Lit(P)$, deciding whether there exists a $\{i, p\}$-preferred answer set M such that $l \in M$ is $\mathbf{\Sigma_2^P}$-complete.*

Proof. Inclusion: We can first guess M such that $l \in M$ and verify that M is an answer set. Then by Theorem 2 we can use an **NP**-oracle to verify that M is $\{i, p\}$-preferred.

Hardness: Given a 3-SAT-instance S and a literal l, it is $\mathbf{\Sigma_2^P}$-hard to decide whether l is true in a minimal model of S [3]. We construct a *LPOD* $t(S)$ such that l is true in a $\{i, p\}$-preferred answer set of $t(S)$ iff l is true in a minimal model of S.

$$t(S) = \{\leftarrow \bar{A}_1, \bar{A}_2, \bar{A}_3 \mid A_1 \vee A_2 \vee A_3 \in S\} \tag{9}$$

$$\cup \{\neg a \times a \mid a \in At(S)\} \tag{10}$$

Now M is a preferred answer set of $t(S)$ iff it is a minimal model of S. We can see this by noting that the rules of the form (10) generate all possible truth valuations for atoms of S and all rules (9) are satisfied whenever all clauses are.

Suppose that M is a minimal model of S and that there exists an answer set M' such that $M' >_{i,p} M$. By (10) this implies that there exists an atom a such that $a \in M$ and $a \notin M'$, and there does not exist an atom b such that $b \in M'$ and $b \notin M$. Thus, $M' \subset M$. However, this is a contradiction since M' is also a model of S and we assumed that M is minimal.

Next, suppose that M is a $\{i, p\}$-preferred answer set of $t(S)$. This implies that there is no answer set M' such that $\exists a \in At(S) : a \in M \wedge a \notin M'$. Thus, there is no $M' \subset M$ and M is a minimal model of S.

Theorem 4. *Given an LPOD P and a literal $l \in Lit(P)$, deciding whether there exists a c-preferred answer set M such that $l \in M$ is in $\mathbf{\Delta_2^P}$.*

Proof. Each answer set M of P induces a tuple $(|M^1|, \ldots, |M^d|)$ where $|M^k|$ denotes the number of rules satisfied to the degree k by M. We can see from Definition 5 that each c-preferred answer set induces the same tuple, $(c^1, \ldots, c^d)$. We can find the correct value of c^1 using $O(\log r)$ adaptive **NP**-oracle queries where r is the number of rules in P. We add to P rules to force $|M^1|$ to be within given upper and lower bounds (u, l) and then perform a binary search to narrow the bounds. The bound rules may be expressed using the notation of Fig. 3 as follows:

$$\begin{aligned} &\leftarrow s\text{-}card(1, N), N < l, degree(N) \\ &\leftarrow s\text{-}card(1, N), N > u, degree(N) \end{aligned}$$

After establishing c^i, we proceed to establish c^{i+1}. Since both r and d are linear with respect to the size n of P, we can find out all c^i using $O(n \log n)$ **NP**-queries. Finally, we add $\leftarrow$ not l and issue one more query to see whether l is true in a c-preferred answer set.

6 An Application: Configuration Management

To illustrate *LPOD*s we want to briefly discuss an application in configuration management. Answer set programming techniques seem to be suitable for modeling many different configuration domains [8, 6]. During a configuration process we have often more than one alternative way to satisfy a requirement. For example, a workstation may have more than one possible email client.

We now consider two examples that show how *LPOD*s can be used to model several kinds of different preference criteria in Linux configuration domain. First, there are usually several available versions for any given software package. In most cases we want to install the latest version, but sometimes we have to use an older one. We can handle these preferences by defining a new atom for each different version and then demanding that at least one version should be selected if the component is installed. For example, the following rule defines that there are three versions of emacs available:

$$\mathit{emacs}\text{-}21.1 \times \mathit{emacs}\text{-}20.7.2 \times \mathit{emacs}\text{-}19.34 \leftarrow \mathit{emacs}$$

Second, a component may have also different variants. For example, most programming libraries come in two versions: a normal version containing only files that are necessary to run programs that are linked against the library, and a developer version with header or class files that allow a developer to create new applications. Now, a common user would prefer to have the normal variant while a programmer would prefer the developer version. We may model these preferences in the following way:

$$\begin{aligned} \mathit{libc6} \times \mathit{libc6\text{-}dev} \leftarrow & \mathit{need\text{-}libc6}, \text{not } \mathit{c\text{-}developer} \\ \mathit{libc6\text{-}dev} \times \mathit{libc6} \leftarrow & \mathit{need\text{-}libc6}, \mathit{c\text{-}developer} \ . \end{aligned}$$

There are also many other possible preference criteria in the configuration domain. It is not at all clear how they should be combined into one comprehensive preference structure, and further work into this direction is needed.

7 Conclusion

In this paper we show how *LPOD*s can be implemented using ASP solvers for non-disjunctive programs with a two-step approach. We first create a disjunction-free generator program that has the same answer sets as P, and use the ASP solver to find a candidate answer set S. Next we use a disjunction-free tester program to check whether S is preferred or not. Since the tester is based on a declarative representation of the preference criterion it is easy to switch between different notions of preference, or to define new ones.

Further contributions of this paper are a Pareto-style preference criterion which, to the best of our knowledge, has not been used in prioritized non-monotonic reasoning before, a combination of ordered disjunctions with preferences among rules, and several complexity results. For a discussion of related approaches to logic programming with context dependent priorities see [1].

In future work we want to study more general qualitative decision making problems. In such settings it is not always sufficient to consider the most preferred answer sets only since this amounts to an extremely optimistic view about how the world will behave (this view is sometimes called wishful thinking). As is well-known in decision theory, for realistic models of decision making it is necessary to clearly distinguish what is under the control of the agent (and thus may constitute the agent's decision) from what is not.

In answer set programming this can be done by distinguishing a subset of the literals as decision literals. *LPOD*s can be used to describe possible actions or decisions and their consequences, states of the world and desired outcomes. Based on the preference ordering on answer sets an ordering on possible decisions can be defined based on some decision strategy. We plan to work this out in more detail in a separate paper.

Acknowledgements. We would like to thank Tomi Janhunen for helpful comments. The second and third author thank Academy of Finland (project 53695) and HeCSE for financial support.

References

1. G. Brewka. Logic programming with ordered disjunction. In *Proc. 18th National Conference on Artificial Intelligence, AAAI-2002.* Morgan Kaufmann, 2002.
2. G. Brewka, S. Benferhat, and D. Le Berre. Qualitative choice logic. In *Proc. Principles of Knowledge Representation and Reasoning, KR-02*, pages 158–169. Morgan Kaufmann, 2002.
3. T. Eiter and G. Gottlob. Propositional circumscription and extended closed-world reasoning are Π_2^P-complete. *Theoretical Computer Science*, 114:231–245, 1993.
4. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
5. Tomi Janhunen, Ilkka Niemelä, Patrik Simons, and Jia-Huai You. Unfolding partiality and disjunctions in stable model semantics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 7th International Conference*, pages 411–419. Morgan Kaufmann Publishers, April 2000.
6. Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer Academic Publishers, 2000.
7. C. Sakama and K. Inoue. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, 13:145–172, 1994.
8. T. Soininen. *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks.* PhD thesis, Helsinki University of Technology, Finland, 2000.