# Logic Programs with Ordered Disjunction

Gerhard Brewka* Ilkka Niemelä† Tommi Syrjänen†

December 19, 2003

**Abstract**

Logic programs with ordered disjunction (*LPOD*s) contain a new connective which allows representing alternative, ranked options for problem solutions in the heads of rules: $A \times B$ intuitively means that 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. 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 and possible applications.

Keywords: logic programs, answer set semantics, priorities, qualitative preferences

## 1 Introduction

The ability to express preferences among potential problem solutions is essential for many applications. Numerous techniques, both numerical and qualitative, have been investigated in AI and operations research, and this special issue gives an excellent overview of some of the current work in this area.

Our interest in this paper is on qualitative preferences. Classical approaches to decision making assume a complete specification of numerical utility and probability functions. Given these functions maximum expected utility is used to select among the possible actions or choices. However, in many cases these functions are difficult to obtain, and users are unwilling to come up with quantified specifications of their preferences. This is the main motivation behind approaches which are based on qualitative preference statements rather than numerical functions.

In many practical cases the qualitative preferences we want to express are preferences among different options or alternatives. In logic, alternatives are described using disjunction. It seems a natural idea to "enhance disjunction" and to use the order in which disjuncts are written down to express what is preferred. For instance, the disjunction

$$cinema \vee pub \vee tv$$

may describe different available possibilities how to spend a free evening. If we use ordered disjunction instead we can additionally express a preference order among the options:

$$cinema \times pub \times tv$$

intuitively says that if possible I would like to go to the *cinema*, if this is not possible then I go to the *pub*, if this is also not possible I will watch *tv*.

Ordered disjunction was first proposed in Brewka et al. (2002) where a propositional logic called Qualitative Choice Logic (*QCL*) was introduced. The logic contains the new connective $\times$. The semantics of the logic is based on degrees of satisfaction of a formula in a classical model. The degrees, intuitively, measure disappointment and induce a preference relation on models. Consequence is defined in terms of most preferred models.

*Universität Leipzig, Institut für Informatik, Augustusplatz 10-11, 04109 Leipzig, Germany, brewka@informatik.uni-leipzig.de

†Helsinki University of Technology, Dep. of Computer Science and Engineering, Laboratory for Theoretical Computer Science, P.O.Box 5400, FIN-02015 HUT, Finland

In this paper we want to combine ideas underlying $QCL$ with logic programming. More precisely, we want to investigate logic programs based on rules with ordered disjunction in the heads. We call such programs logic programs with ordered disjunction (*LPOD*s).

The semantical framework of our investigation is that of answer set semantics (Gelfond and Lifschitz, 1991). Answer set semantics describes the meaning of a logic program $P$ in terms of sets of literals. The exact definition of answer sets depends on the kind of rules used in $P$, yet two properties are always required: answer sets are closed under the rules of $P$, and they are grounded in $P$, that is, each literal has a derivation using "applicable" rules from $P$. Answer set programming has become quite popular for knowledge representation in Artificial Intelligence (AI). There are several reasons for this popularity:

1. Logic programs are expressive enough to model many typical knowledge representation problems in AI. In particular, the availability of default negation in the bodies of rules makes it possible to represent defeasible information.

2. Many problems in reasoning about actions, planning, diagnosis, belief revision and product configuration have elegant formulations as logic programs so that models of programs, rather than proofs of queries, describe problem solutions (Lifschitz, 2002; Soininen, 2000; Baral, 2003).

3. The semantics of answer sets (stable models, for programs without classical negation) is intuitive and avoids the pitfalls of resolution-based systems like Prolog. For instance, it is independent of the order in which rules are written and correctly handles loops.

4. In the same time, the syntax of logic programs is restrictive enough to allow for fast implementations and several highly efficient answer set solvers have been developed. Most advanced among them are *Smodels* (Niemelä and Simons, 1997) and *dlv* (Eiter et al., 1998).

An *LPOD* $P$ may have zero, one, or many answer sets. We define a preference relation over the set of all answer sets of $P$ and we say that an answer set $S$ is preferred if there exists no answer set $S'$ that is better than $S$. It is possible that $P$ has more than one preferred answer set. These cases happen when there are two or more conflicting preferences that are equally important in a sense that will be explained later.

Our approach has two nice properties:

1. If an *LPOD* $P$ has at least one answer set, then at least one of them is preferred.

2. If $S_1$ and $S_2$ are answer sets of $P$, then it is possible to decide if $S_1$ is preferred to $S_2$ (or vice versa) by comparing $S_1$ and $S_2$ directly.

The first property ensures that we always get an answer if there exists one. The preferences act as optional constraints that select a best answer from the set of all solutions. If it happens so that all our preferences cannot be met, then we are willing to accept that and get an answer that satisfies the rest of preferences as well as possible.

The second property makes it easier to generate preferred answer sets since we do not necessarily have to construct the whole preference relation before we can identify its minima. This point will be expanded in Section 5.

The rest of the paper is organized as follows. In the next section we recall the basic notions underlying the syntax and semantics of answer sets. This section is included to make the paper self-contained and can be skipped by readers familiar with these notions. Section 3 introduces the syntax of *LPOD*s and defines the corresponding notion of answer sets. Section 4 discusses several alternative preference relations on answer sets which can be obtained based on the satisfaction degrees of rules. In Section 5 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 the constantly improving performance of ASP solvers for standard logic programs such as *Smodels* and *dlv*. The implementation is based on 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. Section 6 discusses the complexity of reasoning tasks based on *LPOD*s. Section 7 compares *LPOD*s to some other preference formalisms. Some possible application domains are described in Section 8.

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.

## 2 Answer sets

In this section we recall the definition of answer sets as introduced by Gelfond and Lifschitz (1991). We consider extended logic programs which have two kinds of negation, classical negation $\neg$ and default negation not. Intuitively, not $a$ is true whenever there is no reason to believe $a$, whereas $\neg a$ requires a proof of the negated literal. An *extended logic program* (program, for short) $P$ is a finite collection of rules $r$ of the form

$$c \leftarrow a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m \tag{1}$$

where the $a_i, b_j$ and $c$ are classical literals, i.e., either positive atoms or atoms preceded by the classical negation sign $\neg$. We denote by $\mathit{head}(r)$ the head $c$ of rule $r$ and by $\mathit{body}(r)$ the body $a_1, \ldots, a_n$, not $b_1, \ldots$, not $b_m$ of the rule. We will call $a_1, \ldots, a_n$ the *prerequisites* of the rule and use $\mathit{pre}(r)$ to denote the set of prerequisites of $r$.

We say a rule $r$ of the form (1) is *defeated by a literal* $\ell$, if $\ell = b_i$ for some $i \in \{1, \ldots, m\}$, and we say it is *defeated by a set of literals* $X$, if $X$ contains a literal that defeats $r$. Moreover, a rule $r$ is applicable in $X$ whenever it is not defeated by $X$ and its prerequisites are in $X$. We denote this condition by $X \models \mathit{body}(r)$. A rule $r$ is satisfied by $X$ (denoted $X \models r$) if $\mathit{head}(r) \in X$ or if $X \not\models \mathit{body}(r)$. A set of literals $X$ is *consistent* if for all atoms $a \in X$ implies that $\neg a \notin X$.

An answer set of a program $P$ is a set of literals $S$ satisfying two conditions:

1. if $r \in P$ is applicable in $S$, then $r$ is applied, that is, $\mathit{head}(r) \in S$, and
2. all literals in $S$ have a non-circular derivation using only rules undefeated by $S$.

We can make this precise as follows:

**Definition 1** *Let $P$ be an extended logic program, and let $X$ be a set of literals. The $X$-reduct of $P$, denoted $P^X$, is the collection of rules resulting from $P$ by*

1. *deleting each rule which is defeated by $X$, and*
2. *deleting all default negated literals from the remaining rules.*

This construction is often called Gelfond-Lifschitz reduction, after its inventors.

**Definition 2** *Let $R$ be a collection of rules without default negation. Then, $\mathit{Cn}(R)$ denotes the smallest set $S$ of literals such that*

1. *$S$ is closed under $R$, i.e., for any rule $c \leftarrow a_1, \ldots, a_n$ in $R$, if $a_1, \ldots, a_n \in S$, then $c \in S$; and*
2. *$S$ is logically closed, i.e., either $S$ is consistent or $S = \mathit{Lit}(R)$, the set of all literals.*

**Definition 3** *Let $R$ be a collection of rules. Define an operator $\gamma_R(X)$ on the set literals as follows:*

$$\gamma_R(X) = \mathit{Cn}(R^X)$$

*Then, a set $S$ of literals is an answer set of $R$ iff $S = \gamma_R(S)$.*
*The collection of answer sets of $R$ is denoted by $AS(R)$.*

It turns out that many problems can be formulated in a way such that answer sets are (or contain) the solutions. This style of representing problems is called answer set programming (ASP) nowadays (Dimopoulos et al., 1997; Niemelä, 1999; Marek and Truszczyński, 1999; Balduccini and Gelfond, 2003). A good example is graph colouring. The task is to assign a number of different colours (3 in our example) to the nodes in a graph such that neighbouring nodes get different colours. Using rule schemata we can represent this as follows:

$$\begin{aligned} col(X, red) &\leftarrow node(X), \text{not } col(X, blue), \text{not } col(X, green) \\ col(X, blue) &\leftarrow node(X), \text{not } col(X, red), \text{not } col(X, green) \\ col(X, green) &\leftarrow node(X), \text{not } col(X, blue), \text{not } col(X, red) \\ &\leftarrow col(X, C), col(Y, C), edge(X, Y), X \neq Y \end{aligned}$$

Here a rule $\leftarrow body$ with empty head is an abbreviation for

$$new \leftarrow \text{not } new, body$$

where $new$ is a new atom not appearing elsewhere in the program. The effect of the rule is that all answer sets satisfying body are eliminated. Such rules are also called constraints.

It is not difficult to see that the answer sets generated by this program contain colour assignments for each node satisfying the requirements for solutions of the colouring problem.

## 3 Logic programs with ordered disjunction

Logic programs with ordered disjunction are an extension of programs with two kinds of negation (default and strong negation) (Gelfond and Lifschitz, 1991). 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. As usual we omit $\leftarrow$ whenever $m = 0$ and $k = 0$, that is, if the rule is a fact. We say that a rule is satisfied if either one of literals $C_i$ is true or if some literal in the body is false. We use the notations $At(P)$ and $Lit(P)$ to denote the sets of atoms and literals occurring in an *LPOD* $P$, respectively.

Before defining the semantics of *LPOD*s a few observations are in order. As already mentioned in the introduction we want to use the ranking of literals in the heads of rules to select some of the answer sets of a program as the preferred ones. But what are the answer sets of a program among which to make this selection?

Since ordered disjunction is a particular prioritized form of disjunction it seems like a natural idea to base the semantics of *LPOD*s on one of the standard semantics for disjunctive logic programs, for instance, the one presented by Gelfond and Lifschitz (1991).

Unfortunately, this doesn't work. The problem is that most of the semantics for disjunctive logic programs have minimality built in. For instance, according to Gelfond and Lifschitz, $S$ is an answer set of a disjunctive logic program $P$ iff $S$ is a minimal set of literals which is logically closed, and closed under the $S$-reduct of $P$. In this approach answer sets are minimal: if $S_1$ and $S_2$ are answer sets of a disjunctive program $P$ and $S_1 \subseteq S_2$, then $S_1 = S_2$.

Minimality is not always wanted for *LPOD*s. Consider the following two facts:

$$\begin{aligned} A \times B \times C \\ B \times D \end{aligned}$$

The single best way of satisfying both ordered disjunctions is obviously to make $A$ and $B$ true, that is, we would expect $\{A, B\}$ to be the single preferred answer set of this simple *LPOD*. However, since $B$ is sufficient to satisfy both disjunctions, the set $\{A, B\}$ is not even an answer set of the corresponding disjunctive logic program (where $\times$ is replaced by $\vee$) according to the semantics of Gelfond and Lifschitz (1991): the built in minimality precludes sets containing both $A$ and $B$ from consideration.

We thus have to use a semantics which is not minimal. Indeed, there is such a semantics, the possible models semantics proposed by Sakama and Inoue (1994). It is based on so-called split programs, that is, disjunction free programs which contain arbitrary subsets of single head rules obtained from disjunctive rules by deleting all but one alternative in the head.

Unfortunately, also this semantics is inadequate, this time for opposite reasons: it admits too many literals in answer sets. Consider the disjunctive logic program

$$A \vee B \vee C$$

There are seven split programs corresponding to the nonempty subsets of the literals of the fact. The split program containing the facts $A, B, C$ generates the possible model where $A, B, C$ is true.

Let us replace disjunction by ordered disjunction in this formula. According to our intuitive discussion we want to read the rule as "if possible $A$, if this is not possible then $B$, and if also $B$ is not possible then $C$". Under this reading models containing more than one of the literals in the head do not seem justified on the basis of a single rule (they may be justified by different rules, though).

For this reason we will not allow cases where a single rule of the original program gives rise to more than one rule in the split program. There is a further complication. Consider the program:

$$\begin{array}{r} A \times B \times C \\ A \end{array}$$

We do not want to obtain $\{A, B\}$ as an answer set from the split program consisting of these two atomic facts since again this does not correspond to the intuitive reading of the first rule ($B$ only if $A$ is not possible). We therefore have to use slightly more complicated rules in split programs.

Our definition of split programs for *LPOD*s thus 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 4** *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 5** *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.*

**Example 1** *Let $P$ consist of the rules*

$$A \times B \leftarrow \text{not } C \tag{2}$$
$$B \times C \leftarrow \text{not } D \tag{3}$$

*We obtain four 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 6** *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 three 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 (3) and $\{B\}$ the second best option for (2). We therefore introduce the notion of a preferred answer set in the next section.

There is also an alternative way to characterize answer sets of *LPOD*s by defining their reducts directly without using split programs. The basic idea is that the reduct of a rule $r = c_1 \times \cdots \times c_n \leftarrow \mathit{body}$ with respect to a set $X$ of literals is a rule $c_j \leftarrow \mathit{pre}(r)$ where $c_j$ is the lowest-ranking head atom that is present in $X$ and $\mathit{pre}(r)$ the set of positive literals in the rule body. Thus, we use each rule of the program to support the best-ranked atom in its head that is present in the candidate answer set.

**Definition 7** *Let $r = C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \text{not } B_1, \ldots, \text{not } B_k$ be a rule and $X$ a set of literals. Then, the* $\times$-reduct $r_\times^X$ *of $r$ is defined as follows:*

$$\begin{aligned} r_\times^X = \{\ & C_i \leftarrow A_1, \ldots, A_m \mid C_i \in X \textit{ and} \\ & X \cap \{C_1, \ldots, C_{i-1}, B_1, \ldots, B_k\} = \emptyset\}\ . \end{aligned} \tag{4}$$

*Let $P$ be an LPOD. The $\times$-reduct $P_\times^X$ of $P$ is defined as follows:*

$$P_\times^X = \bigcup_{r \in P} r_\times^X\ . \tag{5}$$

**Proposition 1** *Let $P$ be an LPOD and $S$ a set of literals. Then, $S$ is an answer set of $P$ iff the following three conditions hold:*

1. *$S = Cn(P_\times^S)$,*
2. *$S$ is consistent, and*
3. *$S$ satisfies every rule $r \in P$.*

*Proof.*
($\Leftarrow$) Let $S$ be a set of literals satisfying the three conditions of the proposition. Then we can create a split program $P_s$ that generates $S$ by choosing for each rule $r = C_1 \times \cdots \times C_n \leftarrow \mathit{body}$ the option $r^i$ where

$$i = \begin{cases} \min(\{i \mid C_i \in S \text{ and } \forall j < i : C_j \notin S\}), & \text{if } \{C_1, \ldots, C_n\} \cap S \neq \emptyset \\ 1, & \text{otherwise}\ . \end{cases}$$

By construction $P_\times^S \subseteq P_s^S$ holds implying $\mathrm{Cn}(P_\times^S) \subseteq \mathrm{Cn}(P_s^S)$. We show that $\mathrm{Cn}(P_s^S) \subseteq \mathrm{Cn}(P_\times^S)$ holds by establishing that $S$ is closed under $P_s^S$. This implies $\mathrm{Cn}(P_s^S) \subseteq S = \mathrm{Cn}(P_\times^S)$. Consider a rule $r = C_1 \times \cdots \times C_n \leftarrow \mathit{body}$ for which $C_i \leftarrow \mathit{pre}(r)$ in $P_s^S$. If $S \models \mathit{pre}(r)$, then by Condition 3 there is some $C_j \in S$. Then $C_i \leftarrow \mathit{pre}(r) \in P_s^S$ implies $C_i \in S$. Thus, $S$ is closed under $P_s^S$ and $S = \mathrm{Cn}(P_s^S) = \mathrm{Cn}(P_\times^S)$. Since $S$ is consistent, it is an answer set of $P$.
($\Rightarrow$) Let $S$ be an answer set of $P$. Then, there exists a split program $P_s$ such that $S = \mathrm{Cn}(P_s^S)$. Clearly, $S$ is consistent and satisfies all rules in $P$. We show that $S = \mathrm{Cn}(P_\times^S)$. By construction $S$ is closed under $P_\times^S$. Consider some $S' \subseteq S$ also closed under $P_\times^S$. We prove that $S'$ is closed under $P_s^S$ implying that $S = \mathrm{Cn}(P_s^S) \subseteq S'$. This shows that $S = \mathrm{Cn}(P_\times^S)$ holds.

Consider a rule $r = C_1 \times \cdots \times C_n \leftarrow \mathit{body}$ for which $C_i \leftarrow \mathit{pre}(r) \in P_s^S$. Then $\{C_1, \ldots, C_{i-1}\} \cap S = \emptyset$. If $S' \models \mathit{pre}(r)$, then $S \models \mathit{pre}(r)$ and, hence, $C_i \in S$ as $S = \mathrm{Cn}(P_s^S)$. But then $C_i \leftarrow \mathit{pre}(r) \in P_\times^S$ and, thus, $C_i \in S'$. This shows that $S'$ is closed under $P_s^S$. $\square$

proposition. that if $M \vDash \mathit{body}$, then the reduct of $r^i$ and the $\times$-reduct of $r$ are $Cl(P_\times^M)$. Thus answer set of $P$.

**Example 2** *Consider again the program $P$ that was presented in Example 1:*

$$A \times B \leftarrow \text{not } C$$
$$B \times C \leftarrow \text{not } D$$

*Now, given the set $S_1 = \{A, B\}$ the reduct $P_\times^{S_1}$ is:*

$$A$$
$$B \;\; .$$

*Since $Cn(P_\times^{S_1}) = \{A, B\}$, $S_1$ is an answer set. On the other hand, the set $S_2 = \emptyset$ is not an answer set even though $P_\times^{\emptyset} = \emptyset$ and $Cn(\emptyset) = \emptyset = S_2$ since it satisfies neither of the rules.*

# 4 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 8** *Let $S$ be an answer set of an LPOD $P$. Then $S$ satisfies the rule*

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

- *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 possible degree 1 is obtained. The degree of $r$ in $S$ will be denoted 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 9** *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 Brewka (2002):

**Definition 10** *Let $S_1$ and $S_2$ be answer sets of an LPOD $P$. Then $S_1$ 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 particular, inclusion-preference may depend on scaling. Consider the following example about picking a hotel for a conference.[1] You prefer to be in walking distance. Moreover, you prefer a 3* hotel over a 2* hotel. This can be represented as

$$walking \times \neg walking \tag{6}$$
$$3^* \times 2^* \tag{7}$$

[1] The example and the Pareto criterion was suggested to us by Harri Haanpää.

Now assume the only available 3* hotel, say $hotel_1$, is not in walking distance, and the hotel in walking distance, say $hotel_2$, is 2*. Neither of the options is preferred to the other since each of them provides only the second best option for one of the ordered disjunctions. Now assume we represent unsatisfiable better options (unsatisfiable since our university does not refund luxury accommodation). Let us replace (7) with

$$4^* \times 3^* \times 2^* \tag{7'}$$

and add $\neg 4^*$. This has the effect that the walking distance becomes more important and $hotel_2$ is the preferred option. To avoid effects of this kind one can use the following Pareto criterion:

**Definition 11** *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:

**Proposition 2** *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 12** *A set of literals $S$ is a $\kappa$-preferred (where $\kappa \in \{c, i, p\}$) 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 $S' >_\kappa S$.*

The different preference criteria can be illustrated using a prioritized variant of the graph colouring problem. Assume someone prefers red nodes over blue nodes and blue nodes over green ones. The prioritized colouring problem can be represented as follows:

$$col(X, red) \times col(X, blue) \times col(X, green) \leftarrow node(X) \tag{8}$$

$$\leftarrow col(X, C), col(Y, C), node(X), node(Y), edge(X, Y), X \neq Y \tag{9}$$

We can describe preference among the answer sets of the example, given one of the three criteria, as follows: $S$ is preferred over $S'$ if

| | |
|---|---|
| $>_p$: | at least one node has a more preferred colour in $S$ and<br>no node has a less preferred colour in $S$. |
| $>_i$: | the red nodes in $S'$ are a proper subset of the red nodes in $S$, or<br>$S$ and $S'$ have the same red nodes and the blue nodes in $S'$ are<br>a proper subset of the blue nodes in $S$. |
| $>_c$: | more nodes are red in $S$ than in $S'$, or<br>as many nodes are red in $S$ as in $S'$ and more are blue in $S$. |

Given a particular preference criterion $\kappa$, we say a literal $l$ is a conclusion of an *LPOD* $P$ iff $l$ is contained in all $\kappa$-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.

*LPOD*s allow us — like normal logic programs — to express incomplete and defeasible knowledge through the use of default negation. In addition, they provide means to represent preferences among intended properties of problem solutions. Moreover, these preferences may depend on the current context. Here is an example about how to spend a free afternoon. You like to go to the beach, but also to the cinema. Normally you prefer the cinema over the beach, unless it is hot (which is the exception in the area where you live, except during the summer). If it is hot, the beach is preferred over the cinema. In summer it is normally hot, but there are exceptions. If it rains, the beach is out of question. This information can be represented using the following rules:

$$cinema \times beach \leftarrow \text{not } hot \tag{10}$$

$$beach \times cinema \leftarrow hot \tag{11}$$

$$hot \leftarrow \text{not } \neg hot, summer \tag{12}$$

$$\neg beach \leftarrow rain \tag{13}$$

Without further information about the weather we obtain the single preferred answer set $S_1 = \{cinema\}$. There is no information that it might be hot, so rule (10) will determine the preferences. We see that $S_1$ satisfies all rules to degree 1.

Now assume the fact *summer* is additionally given. In this case we obtain $S_2 = \{summer, hot, beach\}$ as the single preferred answer set. Again this answer set satisfies all rules to degree 1.

Next assume that, in addition to *summer* also the literal $\neg hot$ is given. The single preferred answer set now is $S_3 = \{summer, \neg hot, cinema\}$. All rules are satisfied to degree 1.

Finally, assume the additional facts are *summer* and *rain*. Now the single preferred answer set (and in fact the single answer set) is

$$S_4 = \{summer, rain, hot, \neg beach, cinema\}.$$

Note that this time it is not possible to satisfy all rules to degree 1: rule (11) is satisfied to degree 2 only. As often in real life, there are situations where the best options simply do not work out.

Sometimes we may want to express that one preference is more important than another. Assume we have

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

where $r_n$ are used as labels of the rules. We obtain two preferred answer sets (independently of the chosen preference criterion), namely $S_1 = \{ice\text{-}cream, tea\}$ and $S_2 = \{coffee, cake\}$. Assume further that we would rather have ice-cream than coffee. Now we would like to 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 13** *Let $S_1$ and $S_2$ be answer sets of an LPOD $P$, $\succ$ a preference relation on the rules of $P$. Then $S_1$ 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 that 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. This way preferences can become context dependent. However, since in this case the information about rule preferences is a part of the generated answer sets, one has to be careful not to loose anti-symmetry. For instance, if we have

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

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

# 5 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 Janhunen et al. (2000) to compute stable models of disjunctive logic programs using *Smodels*. The two programs are:

- A *generator* $G(P)$ that creates all answer sets of $P$; and
- A *tester* $T(P, S)$ that checks whether a given answer set $S$ of $P$ is preferred.

The two programs are run in an interleaved fashion. First, the generator constructs an answer set $S$ of $P$. Next, the tester tries to find an answer set $S'$ that is strictly better than $S$. If there is no such $S'$, we know that $S$ 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 $S'$ directly as the new answer set candidate.

In the programs we use *cardinality constraint literals* (Niemelä and Simons, 2000) to make them more compact. A cardinality constraint is of the form:

$$L \; \{l_1, \ldots, l_n\} \; U \tag{14}$$

where $l_1, \ldots, l_n$ are literals and $L$ and $U$ are integral lower and upper bounds. The intuition is that (14) is true if the number of satisfied literals $l_i$ is between $L$ and $U$, inclusive. Either of the bounds may be left out and in this case the values 0 and $\infty$ are used in the place of the missing bound. In the programs we sometimes define the literals using set notation as in: $\{a(X) : d(X)\}$. The intuition is now that $a(X) : d(X)$ is replaced by the set of all ground atoms $a(x)$ such that $d(x)$ holds.

Even though adding constraint literals to the language of logic programs increases expressivity, it does not increase complexity; determining the existence of a stable model of a ground program stays in **NP** (Niemelä and Simons, 2000).

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 $k$th 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 14** *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 $k$th option of $r$ is defined as follows:*

$$G(r,k) = \{C_k \leftarrow c(r,k), \text{not } C_1, \ldots, \text{not } C_{k-1}, \mathit{body}; \tag{15}$$

$$\leftarrow \text{not } c(r,k), C_k, \text{not } C_1, \ldots, \text{not } C_{k-1}, \mathit{body}\} \tag{16}$$

*The satisfaction translation $S(r)$ is:*

$$S(r) = \{s(r,1) \leftarrow \text{not } c(r,1), \ldots, \text{not } c(r,n)\} \cup \tag{17}$$

$$\{\, s(r,i) \leftarrow c(r,i) \mid 1 \le i \le n\} \tag{18}$$

*The translation $G(r)$ is:*

$$G(r) = \{1 \; \{c(r,1), \ldots, c(r,n)\} \; 1 \leftarrow \mathit{body}\} \cup \bigcup_{k \le n} G(r,k) \cup S(r) \tag{19}$$

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

$$G(P) = \bigcup_{r \in P} G(r) \tag{20}$$

The definition of $G(P)$ is rather straightforward, but the reason for having two rules in $G(r, k)$ may not be clear. The intuition of the second rule is that we must choose to add $C_k$ if no better literal $C_i$, $i < k$, is already present in the model.

**Example 3** *The program $P$ from Example 1 is translated to:*

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

$$
\begin{aligned}
better(I) &\leftarrow s(R,I), o(R,J), I < J, rule(R), degree(I), degree(J) \\
worse(I) &\leftarrow s(R,J), o(R,I), I < J, rule(R), degree(I), degree(J) \\
better &\leftarrow \{worse(I) : degree(I)\}\ 0, better(J), degree(J)
\end{aligned}
$$

Figure 1: The Pareto-preference tester $T_p$

*The three answer sets are* $S_1 = \{A, B, c(1,1), c(2,1), s(1,1), s(2,1)\}$, $S_2 = \{B, c(1,2), c(2,1), s(1,2), s(2,1)\}$, *and* $S_3 = \{C, c(2,2), s(1,1), s(2,2)\}$. *Note that in* $S_3$ *there is no atom* $c(1,k)$ *since the body of the first rule is not satisfied.*

**Proposition 3** *Let* $P$ *be an LPOD.*

1. *If* $S$ *is a consistent answer set of* $G(P)$, *then* $S' = S \cap Lit(P)$ *is an answer set of* $P$.
2. *If* $S$ *is an answer set of* $P$, *then there exists a consistent answer set* $S'$ *of* $G(P)$ *such that* $S = S' \cap Lit(P)$.

*Proof.*

1. Suppose that $S$ is a consistent answer set of $G(P)$. Then, for each atom $C \in S' = S \cap Lit(P)$, there exists a rule of the form (15) where $C$ is the head and some atom $c(r,k) \in S'$ occurs positively in the body. We define $P' = \{r^k \mid c(r,k) \in S'\} \cup \{r^1 \mid \neg\exists k : c(r,k) \in S'\}$. We now see that $P'$ is a split program of $P$ that generates $S$ as its answer set, so $S$ is an answer set of $P$.
2. Let $S$ be an answer set of $P$. Now, for each rule $r = C_1 \times \cdots \times C_n \leftarrow body$ we define $p(S,r) = \{c(r,k), s(r,k) \mid S \models body, C_k \in S, \text{ and } \forall i < k : C_i \notin S\} \cup \{s(r,1) \mid r \in P \text{ and } S \not\models body\}$. Let $S' = S \cup \bigcup_{r\in P} p(S,r)$. By definition of $p(S,r)$, $S'$ satisfies all rules (15)–(18). Finally, (19) is satisfied since exactly one atom $c(r,k)$ was added to $S'$ for each rule $r$ that had its body true. So, $S' \models G(P)$ and hence, $S' \models G(P)^{S'}$. It is straightforward to show that if $S'' \subseteq S'$ and $S'' \models G(P)^{S'}$, then $S'' = S'$. This establishes that $S'$ is an answer set of $G(P)$.

□

Since we have three different optimality criteria, we need three different tester programs. They all consist of a common core $C(P,S)$ that is augmented by case-specific rules $T_c$, $T_i$, or $T_p$. We add a new atom *better* to denote the case when a new answer set $S'$ is strictly better than a given $S$. In the rules that define *better* we also make use of an auxiliary predicate $worse/1$ to signify the cases where $S'$ is in some aspect worse than $S$.

In the construction of $C(P,S)$ we use the notation $max_d(P)$ to denote the maximum number of choices that occur in the head of any rule in $P$:

$$max_d(P) = \max\{n \mid C_1 \times \cdots \times C_n \leftarrow body \in P\}\ .$$

**Definition 15** *Let* $P$ *be an LPOD. Then, the core tester* $C(P,S)$ *is defined as follows:*

$$
\begin{aligned}
C(P,S) = G(P) &\cup \{o(r,k) \leftarrow\mid s(r,k) \in S\} \cup \{rule(r) \leftarrow\mid r \in P\} \\
&\cup \{degree(d) \leftarrow\mid 1 \le d \le max_d(P)\} \\
&\cup \{\leftarrow \text{not } better\}
\end{aligned}
$$

*The* $\kappa$*-preference tester* ($\kappa \in \{c,i,p\}$) $T_\kappa(P,S)$ *is defined as follows:*

$$T_\kappa(P,S) = C(P,S) \cup T_\kappa$$

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 $S$ so that $o(r,k)$ is added as a fact to $T_\kappa(P,S)$ whenever $s(r,k) \in S$.

The $p$-preference tester $T_p$ (Fig. 1) is by far the simplest. It states that $S'$ is better if there exists some rule that has a lower degree of satisfaction in $S'$ and there are no rules with a higher one. The $i$-preference

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

Figure 2: The inclusion preference tester $T_i$

$$
\begin{aligned}
s\text{-}card(I,N) &\leftarrow N\ \{s(R,I) : rule(R)\}\ N, degree(I), num(N) \\
o\text{-}card(I,N) &\leftarrow N\ \{o(R,I) : rule(R)\}\ N, degree(I), num(N) \\
better(I) &\leftarrow s\text{-}card(I,N_1), o\text{-}card(I,N_2), N_1 > N_2, degree(I) \\
worse(I) &\leftarrow s\text{-}card(I,N_1), o\text{-}card(I,N_2), N_1 < N_2, degree(I) \\
better &\leftarrow \{worse(I) : degree(I) \wedge I \leq J\}\ 0, better(J), degree(J) \\
\{num(i) &\leftarrow\ \mid 0 \leq i \leq |P|\}
\end{aligned}
$$

Figure 3: The cardinality preference tester $T_c$

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

**Proposition 4** *Let $P$ be an LPOD and $S$ be a consistent answer set of $G(P)$.*

1. *If $S'$ is an answer set of $T_\kappa(P,S)$ ($\kappa \in \{c,i,p\}$), then $S'' = S' \cap Lit(P)$ is a consistent answer set of $P$ and $S''$ is $\kappa$-preferred to $S$.*

2. *If there exists an answer set $S'$ of $P$ that is $\kappa$-preferred to $S$ ($\kappa \in \{c,i,p\}$), then there exists a consistent answer set $S''$ of $T_\kappa(P,S)$ such that $S' = S'' \cap Lit(P)$.*

*Proof.* (For $p$-preference)

1. Let $S'$ be a consistent answer set of $T_p(P,S)$. Now, $S' \cap Lit(G(P))$ is a consistent answer set of $G(P)$ as $G(P) \subset T_p(P,S)$ and for all rules $r \in T_p(P,S) \setminus G(P)$, $head(r) \notin Lit(G(P))$. Thus, by Proposition 3 $S' \cap Lit(P)$ is a consistent answer set of $P$. By definition of $C(P,S)$, $better \in S'$. By construction of $T_p(P,S)$, we see that $better$ is true exactly when $\exists r : deg_{S'}(r) < deg_S(r)$ and $\neg\exists r : deg_{S'}(r) > deg_S(r)$ so $S' \cap Lit(P) >_p S$.

2. If there exists $S' >_p S$, then $S'$ can be generated by the $G(P)$ part of $T(P,S)$ so $S' \cup \bigcup\{p(S',r) \mid r \in P\} \cup \{better\}$ is an answer set of $T_p(P,S)$ where $p(S',r)$ is defined as in proof of Proposition 3.

The cases for $c$- and $i$-preference are analogous. □

The following corollary is immediate from Proposition 4:

**Corollary 1** *Let $P$ be an LPOD and $S$ be a consistent answer set of $G(P)$. Then, $S$ is $\kappa$-preferred ($\kappa \in \{c,i,p\}$) iff $T_\kappa(P,S)$ does not have any consistent answer sets.*

We can handle meta-preferences by modifying the rules of the $worse/1$ predicate. We define a new predicate $excused(r)$ to denote that some more important rule $r'$, $r' \succ r$, is satisfied to a lower degree in $S'$ than it was in $S$ 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. Unfortunately, this approach does not work for the $c$-preference tester. However, in this case we can define integral weights for the different $c(r,k)$ atoms and then use the sums of the weights to define the $s\text{-}card/2$ and $o\text{-}card/2$ predicates.

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

$$
\begin{aligned}
better(R) &\leftarrow s(R,I), o(R,J), I < J, rule(R), degree(I), degree(J) \\
worse(R) &\leftarrow s(R,J), o(R,I), I < J, rule(R), \text{not } excused(R), degree(I), degree(J) \\
excused(R_1) &\leftarrow R_2 \succ R_1, better(R_2), rule(R_1), rule(R_2) \\
better &\leftarrow \{worse(R_1) : rule(R_1)\}\ 0, better(R_2), rule(R_2)
\end{aligned}
$$

Figure 4: A meta-preference tester

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`.

# 6 Complexity

In the previous section we defined two extended logic programs that can be interleaved to compute $\kappa$-preferred answer sets. Would it be possible to compute them using a single disjunction-free program? Unfortunately, it turns out to be impossible in the general case (unless the polynomial hierarchy collapses); credulous reasoning on *LPOD*s is $\Sigma^{\mathbf{2}}_{\mathbf{P}}$-complete for $\{i,p\}$-preferences. The $c$-preference is slightly easier computationally and it stays in $\Delta^{\mathbf{2}}_{\mathbf{P}}$. Whether it is $\Delta^{\mathbf{2}}_{\mathbf{P}}$-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 $>_\kappa$ relations are all antisymmetric and well-founded, an *LPOD* $P$ has at least one $\kappa$-preferred answer set if it has any answer sets at all.

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

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

*Proof.* (*Inclusion*) If $S$ is not a $\kappa$-preferred answer set of $P$, then there exists an answer set $S'$ such that $S' >_\kappa S$. We can find $S'$ with one query to an NP-oracle so showing that $S$ is preferred is in **coNP**.

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

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

□

**Theorem 3** *Given an LPOD $P$ and a literal $l \in Lit(P)$, deciding whether there exists a $\{i,p\}$-preferred answer set $S$ such that $l \in S$ is $\Sigma^{\mathbf{2}}_{\mathbf{P}}$-complete.*

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

(*Hardness*) Given a 3-SAT-instance $C$ and a literal $l$, it is $\Sigma^{\mathbf{2}}_{\mathbf{P}}$-hard to decide whether $l$ is true in a minimal model of $C$ (Eiter and Gottlob, 1993). We construct a *LPOD* $t(C)$ such that $l$ is true in a $\{i,p\}$-preferred answer set of $t(C)$ iff $l$ is true in a minimal model of $C$.

$$
\begin{aligned}
t(S) = {} & \{\leftarrow \neg A_1, \neg A_2, \neg A_3 \mid A_1 \vee A_2 \vee A_3 \in C\} && (21) \\
& \cup \{\neg a \times a \mid a \in At(C)\} && (22)
\end{aligned}
$$

Now $S$ is an answer set of $t(C)$ iff it is a model of $C$. We can see this by noting that the rules of the form (22) generate all possible truth valuations for atoms of $C$ and all rules (21) are satisfied whenever all clauses are.

Suppose that $S$ is a minimal model of $C$ and that there exists an answer set $S'$ such that $S' >_{i,p} S$. By (22) this implies that there exists an atom $a$ such that $a \in S$ and $a \notin S'$, and there does not exist an atom $b$ such that $b \in S'$ and $b \notin S$. Thus, $S' \subset S$. However, this is a contradiction since $S'$ is also a model of $C$ and we assumed that $S$ is minimal.

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

By this result we see that *LPOD*s under $i$- and $p$-preference are in the same complexity class as disjunctive logic programs. It turns out that there exists a simple translation from disjunctive programs to *LPOD*s. Each disjunction of the form:

$$C_1 \vee \cdots \vee C_n \leftarrow \mathit{body}$$

is replaced by the following $3n+1$ rules:

$$\begin{aligned} &\leftarrow \text{not } C_1, \ldots, \text{not } C_n, \mathit{body} \\ C_i &\leftarrow \text{not } C_i', \mathit{body} \\ C_i' &\leftarrow \text{not } C_i \\ C_i' \times C_i &\leftarrow \mathit{body} \ . \end{aligned}$$

Now, $S$ is an $\{i,p\}$-preferred answer set of the *LPOD* exactly when $S \cap \mathit{Lit}(P)$ is an answer set of the corresponding disjunctive program $P$.[2]

**Theorem 4** *Given an LPOD $P$ and a literal $l \in \mathit{Lit}(P)$, deciding whether there is a c-preferred answer set $S$ such that $l \in S$ is in $\Delta_{\mathbf{P}}^{\mathbf{2}}$.*

*Proof.* Each answer set $S$ of $P$ induces a tuple $(|S^1|, \ldots, |S^d|)$ where $|S^k|$ denotes the number of rules satisfied to the degree $k$ by $S$. We can see from Definition 9 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 $\log n$ adaptive **NP**-oracle queries where $n$ is the number of rules in $P$. We add to $P$ rules to force $|S^1|$ 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:

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

After establishing $c^i$, we proceed to establish $c^{i+1}$. Since both $n$ and $d$ are linear with respect to the size 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. □

# 7 Comparison to Other Formalisms

There are numerous papers introducing preferences to logic programming. For an overview of some of them see the discussion in Brewka and Eiter (1999) or the more recent in Schaub and Wang (2001). The approaches can roughly be characterized along two dimensions: (1) preference among rules vs. preference among atoms/literals, and (2) fixed vs. context-dependent preference.

Examples of fixed rule preference are Brewka and Eiter (1999), Laenens and Vermeir (1990), Buccafurri et al. (1996), and Buccafurri et al. (1998), whereas Sakama and Inoue (2000) introduce fixed preferences among literals.[3] Their approach will be discussed below in more detail.

Only few proposals allow for context dependent preferences. Context dependent rule preferences are discussed for instance in Brewka (1996), Brewka and Eiter (1999), Gelfond and Son (1998), Delgrande et al. (2003), and Grosof (1999). The representation of the preferences in these papers is based on the

[2]Note that $i$-preference and $p$-preference coincide whenever there are not more than 2 ordered disjuncts in each rule head.

[3]The authors also show how conditional preferences and preferences among rules can be modelled using additional symbols in the language.

introduction of names for rules and the explicit representation of the preference relation among rules in the logical language. Two further examples (Brewka, 1996; Brewka and Eiter, 1999) use a quite sophisticated reformulation of the central semantic notion (answer set, extension, etc.) with a self-referential flavour. Compilation techniques which transform prioritized programs to non-prioritized ones are used in Delgrande et al. (2003), Grosof (1999), and Gelfond and Son (1998). They are based on the use of meta-predicates in the logical language which explicitly control the applicability of rules.

Nothing like this is necessary in our approach which belongs to the category of context dependent preferences among literals. All we have to do is use the degree of satisfaction of a rule to define a preference relation on answer sets directly.

In the rest of this section we discuss the relationship between *LPOD*s and two different semantics for expressing preferences, namely *weak constraints* (Buccafurri et al., 2000) and *prioritized logic programs* (Sakama and Inoue, 2000). The former are of interest for our comparison because they are implemented in the *dlv* system and widely used, the latter because they define preferences among literals - as our approach.

## 7.1 Weak Constraints

A *weak constraint* $c$ is of the form:

$$\Leftarrow body \ . \tag{23}$$

The constraint $c$ is *violated* if *body* is true. Intuitively, an answer set is $w$-preferred[4] if it minimizes the number of violated weak constraints. For example, consider the normal logic program $P$:

$$\begin{array}{r} A \leftarrow \text{not } B \\ B \leftarrow \text{not } A \\ \Leftarrow B \ . \end{array}$$

This program has two answer sets, $S_1 = \{A\}$ and $S_2 = \{B\}$. As $S_1$ does not violate the only weak constraint while $S_2$ violates it, $S_1$ is the only $w$-preferred answer set.

We can translate a normal logic program $P$ with weak constraints into an *LPOD* $P'$ by replacing every constraint $c$ of the form (23) by the following two rules:

$$\begin{array}{l} sat_c \times \neg sat_c \leftarrow \\ \quad \leftarrow sat_c, body \ . \end{array}$$

Here $sat_c$ is a new atom that denotes that the constraint $c$ is not violated. The first rule says that we prefer the constraint to be satisfied and the second rule ensures that it is not satisfied if *body* is true. We see that now $w$-preference corresponds to $c$-preference; the number of violated weak constraints is exactly the number of rules satisfied to the second degree.

As some constraints may be more important than others, Buccafurri et al. (2000) et.al. provide a method for assigning different weights to different constraints. For each program $P$, we have a finite sequence $W = \langle W_1, \ldots, W_n \rangle$ of finite sets of weak constraints. The intuition is that if $w \in W_i$ and $w' \in W_j$ where $i < j$, then it is more important to satisfy $w'$ than $w$. Formally, each set $W_i$ has a positive weight $f(W_i)$ assigned to it such that:

$$\begin{array}{r} f(W_1) = 1 \\ f(W_{i+1}) = f(W_i) \cdot |W| + 1 \end{array}$$

where $|W|$ denotes the total number of weak constraints in $P$. An answer set $S$ is preferable to $S'$ if the sum of weights of violated constraints in $S$ is smaller than the sum in $S'$. Thus, satisfying a single constraint belonging to $W_{i+1}$ is more important than satisfying all constraints in $W_i$.

A weak constraint $c \in W_i$ can be translated into following two *LPOD* rules:

$$\begin{array}{c} unsat_n \times \cdots \times unsat_{i+1} \times sat_c \times unsat_{i-1} \times \cdots \times unsat_1 \times \neg sat_c \\ \leftarrow sat_c, body \end{array}$$

[4] Buccafurri et al. (2000) use the notion $\Gamma$-model to denote preferred models.

where all atoms $unsat_j$ are forced to be false in all answer sets by adding strong constraints $\leftarrow unsat_j$. Now, the first rule is satisfied to the degree $n - i + 1$ if the weak constraint is satisfied and to the degree $n + 1$ if it is not satisfied, so we see that the $w$-preference again corresponds to the $c$-preference.

The *dlv* system implements now a more generalized version of weights (Leone et al., 2002) where each individual weak constraint may be assigned a different integral weight. Then, a weak constraint with weight 2 is twice as important as one with the weight of 1. This approach can be emulated in *LPOD*s by adding similar weights to ordered disjunctions and then defining $c$-preference in terms of the sums of weights of rules satisfied to the degree $k$ instead of the number of rules satisfied to the degree.

As normal logic programs with weak constraints belong to the same complexity class $\Delta^{\mathbf{2}}_{\mathbf{P}}$ (Buccafurri et al., 2000) as *LPOD*s under $c$-preference the translation to the other direction also exists. However, we have not found a simple way to construct such a translation. As both $i$- and $p$-preferences of *LPOD*s belong to $\Sigma^{\mathbf{2}}_{\mathbf{P}}$, it is very improbable that we could construct a concise translation for them to normal logic programs with weak constraints.

Similarly, as disjunctive logic programs with weak constraints belong to the complexity class $\Delta^{\mathbf{3}}_{\mathbf{P}}$ (Buccafurri et al., 2000), it is improbable that they can represented using *LPOD*s without an exponential blowup in the size of the translated program. As the complexity of disjunctive programs is $\Sigma^{\mathbf{2}}_{\mathbf{P}}$ even without weak constraints, we see that weak constraints are not necessary for expressing *LPOD*s using disjunctive programs. However, we have not found a simple translation for it. A serious difficulty in defining the translation is that the preferred answer sets of an *LPOD* $P$ are not necessarily minimal.

### 7.2 Prioritized Logic Programs

A *prioritized logic program* ($PLP$) (Sakama and Inoue, 2000) consists of a logic program $P$ and a reflexive and transitive *priority relation* $\Phi \subset Lit(P) \times Lit(P)$. The intuition is that $A \preceq B \in \Phi$ if having $B$ true is at least as important than having $A$ true. The answer sets of $P$ are defined in the usual way, and the relation $\Phi$ is used to define a *pl*-preference relation $\sqsubseteq\ \subseteq AS(P) \times AS(P)$ over all answer sets of $P$ to be the smallest set that fulfills the following three conditions:

1. $S \sqsubseteq S$ for all $S \in AS(P)$;
2. $S_1 \sqsubseteq S_2$ if
$$\begin{aligned}\exists e_2 \in S_2 \setminus S_1 [&\exists e_1 \in S_1 \setminus S_2 \text{ s.t. } e_1 \preceq e_2 \\ &\text{and } \neg\exists e_3 \in S_1 \setminus S_2 \text{ s.t. } e_2 \preceq e_3];\end{aligned}$$
3. if $S_1 \sqsubseteq S_2$ and $S_2 \sqsubseteq S_3$, then $S_1 \sqsubseteq S_3$.

An answer set $S_2$ is preferable to $S_1$ if there exists some element $e_2 \in S_2 \setminus S_1$ that is better than some element $e_1 \in S_1 \setminus S_2$ and $S_1$ does not contain any element that is better than $e_2$. An answer set $S$ is *pl*-preferred[5] if there does not exist an answer set $S' \in AS(P)$ such that $S \sqsubseteq S'$ but not $S' \sqsubseteq S$.

For example, consider the $PLP$ $P$:

$$\begin{aligned} P :\ & \mathit{coffee} \leftarrow \text{not } \mathit{tea} \\ & \mathit{tea} \leftarrow \text{not } \mathit{coffee} \\ \Phi :\ & \mathit{tea} \preceq \mathit{coffee} \end{aligned}$$

Then, there are two answer sets: $S_1 = \{\mathit{tea}\}$ and $S_2 = \{\mathit{coffee}\}$. Since $\mathit{coffee} \in S_2 \setminus S_1$, $\mathit{tea} \in S_1 \setminus S_2$, and $\mathit{tea} \preceq \mathit{coffee}$, the answer set $S_2$ is *pl*-preferred.

Non-disjunctive prioritized logic programs are $\Sigma^{\mathbf{2}}_{\mathbf{P}}$-complete (Sakama and Inoue, 2000) as are *LPOD*s under $i$- and $p$-preference, so we know that they can be translated into each other. However, we have found a simple translation only from *LPOD*s under $p$-preference to $PLP$ programs.

We can translate *LPOD*s under $p$-preference into $PLP$ programs with the help of the translation $G(P)$ that was defined on page 14. If $P$ is an *LPOD*, then its $p$-preferred answer sets correspond to the *pl*-preferred

[5] Sakama and Inoue (2000) use simply the terms *preferred model* and $p$-answer set.

answer sets of the $PLP$ program $\langle G(P), \Phi \rangle$ where the preferences in $\Phi$ are defined over the atoms $s(r,i)$ that encode degrees of satisfactions of ordered disjunctions:

$$\Phi = \{s(r,i) \preceq s(r,j) \mid r \in P \text{ and } i \geq j\} \ .$$

For the $LPOD$ presented in Example 1 we get the priority relation $\Phi = \{s(1,2) \preceq s(1,1), s(2,2) \preceq s(2,1)\}$. Comparing the three answer sets presented in Example 3 we see that $S_2 \sqsubseteq S_1$ as $s(1,1) \in S_1 \setminus S_2$ and $s(1,2) \in S_2 \setminus S_1$. Similarly, $S_3 \sqsubseteq S_1$ so $S_1$ is *pl*-preferred.

A translation to the other direction is more difficult since it is possible that for $PLP$ answer sets $S_1$ and $S_2$, $S_1 \sqsubseteq S_2$ even though $S_1$ and $S_2$ are not directly comparable. For example, if $PLP$ $P$ has three answer sets:

$$\begin{aligned} S_1 &= \{a_2\} \\ S_2 &= \{a_1, b_2\} \\ S_3 &= \{b_1\} \end{aligned}$$

where $a_2 \preceq a_1$, and $b_2 \preceq b_1$, then $S_1 \sqsubseteq S_2$ and $S_2 \sqsubseteq S_3$ so $S_1 \sqsubseteq S_3$ even though $S_1$ and $S_3$ cannot be directly compared. Since $LPOD$s do not have this property, it is clear that it is not easy to construct a direct translation from $PLP$s to $LPOD$s. However, this property also makes it difficult to construct an algorithm for computing *pl*-preferred models. We cannot use a single tester program $T(P,S)$ that is defined analogously to the one presented in Section 5 since an answer set may be preferred even if the tester has an answer set if the $\sqsubseteq$-relation has a cycle.

# 8 Possible Application Areas

In this section we discuss two problem domains that could be suitable for $LPOD$s, configuration management and policy monitoring. In both cases we are interested in finding sets of objects that fulfill some conditions. In configuration management the objects are individual components that together compose the whole system and in policy management we have sets of actions that have to be taken.

## 8.1 Configuration Management

Answer set programming techniques seem to be suitable for modeling many different configuration domains (Soininen, 2000). During a configuration process we have often more than one alternative way to satisfy a requirement. For example, while configuring a workstation there might be several different email reader programs. Now, we would want to express preferences over the different choices. One possible way was presented in Syrjänen (2000) in the context of configuring Linux systems. The semantics is closely related to our $c$-preference but it is strictly less expressive.

We now consider two examples that show how $LPOD$s can be used to model several kinds of different preference criteria in the 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:

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

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:

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

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.

### 8.2 Policy Monitoring

Another possible application is in policy monitoring (Chomicki and Lobo, 2001; Chomicki et al., 2000). Assume we are given a set of action rules specifying which action $a$ is required, possibly under certain additional conditions $C$, when a set of events $\{e_1, \ldots, e_n\}$ occurs. Given a set of events $E$ the problem is to find a set of actions $A$ to be performed in response to $E$. Not all actions may occur together, that is, there is a set of constraints excluding certain combinations of actions. The task now can be defined in two different ways:

- find a maximal set of actions which can be performed in response to $E$, or
- find a maximal set of events such that all actions required by these events can be performed.

It is straightforward to represent the two problems using *LPOD*s. Let us first describe action maximization. The action rules are represented as choice rules of the form

$$a \times \neg a \leftarrow e_1, \ldots, e_n, C$$

and the constraints as standard constraints in answer set programming

$$\leftarrow a_1, \ldots, a_n.$$

Assume the events which occur are represented as atomic facts in the program. Now dependent on the preference criterion chosen preferred answer sets will contain a maximal number (cardinality preference) or an inclusion maximal subset (inclusion preference) of compatible actions required in response to the events which occur.

The second task, event maximization, is not more difficult. We need to distinguish between occurrence of events and handling of events. For each possible event $e$ we add a choice rule

$$\mathit{handle}(e) \times \neg \mathit{handle}(e) \leftarrow e.$$

The action rules become

$$a \leftarrow \mathit{handle}(e_1), \ldots, \mathit{handle}(e_n), C.$$

Constraints remain unchanged. If the events which occur are represented as atomic facts, then the preferred answer sets contain a maximal number, respectively inclusion maximal set, of events which can be handled, together with the necessary actions.

Given the techniques developed in Section 5 it is straightforward to add preferences among the events to be handled.

## 9 Conclusion

In this paper we introduced a new connective to answer set programming. This connective - called ordered disjunction - can be used to represent context dependent preferences in a simple and elegant way. We investigated several criteria for preferences among answer sets based on the satisfaction degrees of rules.

We further showed how *LPOD*s can be implemented using ASP solvers for non-disjunctive programs with a two-step approach. We first translate an *LPOD* $P$ to a disjunction-free program $P'$ possessing the same answer sets. We can thus use an ASP solver to compute answer sets of $P$. A candidate $S$ obtained this way can then be tested for optimality. The tester for $S$ is a disjunction-free program which has no answer set if $S$ is preferred and produces a better answer set otherwise. We can thus iterate until a maximally preferred answer set is reached. 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 combination of ordered disjunctions with preferences among rules, and several complexity results.

Logic programs with ordered disjunction have interesting applications, in particular in design and configuration. In a related paper (Brewka, 2003) possible applications of *LPOD*s for qualitative decision making are discussed. In decision making 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 the future.

In Sect. 7 we discussed related work in logic programming. There is also a large body of literature on preferences in constraint satisfaction. A classical constraint problem is given by a set of variables $V$, a domain $D_{v_i}$ for each variable $v_i$, and a set of constraints specifying conditions for solutions. A solution is an assignment of values from the respective domain to the variables satisfying all constraints. In valued (sometimes also called weighted) constraint satisfaction (Freuder and Wallace, 1992; Fargier et al., 1993; Schiex et al., 1995; Bistarelli et al., 1997) a constraint, rather than specifying hard conditions, yields a ranking of solutions. A global ranking of solutions then is obtained from the rankings provided by the single constraints through some combination rule. In MAX-CSP (Freuder and Wallace, 1992), for instance, constraints assign penalties to solutions and solutions with the lowest penalty sum are preferred. In fuzzy CSP (Fargier et al., 1993) each solution is characterized by the worst violation of any constraint. Preferred solutions are those where the worst violation is minimal. A very general qualitative treatment of prioritized constraints with different preference combination criteria is given in Junker (2002). It is a topic of further research whether the combination rules used in constraint satisfaction have interesting applications in logic programming.

Other topics of interest are hybrid preference models which combine qualitative preferences based on ordered disjunction with numerical preference information, and a generalization of *LPOD*s which allows us to specify partial preferences in the heads of rules.

## Acknowledgements

We would like to thank Tomi Janhunen and the reviewers for helpful comments. The second and the third authors also want to thank Academy of Finland (project no. 53695) and HeCSE for financial support.The first author acknowledges support from DFG (Computationale Dialektik: BR 1817/1-5) and EC (WASP: IST-2001-37004).

## References

M. Balduccini and M. Gelfond. Diagnostic reasoning with a-prolog. *Theory and Practice of Logic Programming*, 3(4-):425–461, 2003.

C. Baral. *Knowledge representation, reasoning and declarative problem solving.* Cambridge University Press, 2003. ISBN 0521818028.

S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 4:19–36, 1996.

G. Brewka. Logic programming with ordered disjunction. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02).* Morgan Kaufmann, 2002.

G. Brewka. Answer sets and qualitative decision making. *Synthese*, 2003. to appear.

G. Brewka, S. Benferhat, and D. Le Berre. Qualitative choice logic. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 8th International Conference, KR-02.* Morgan Kaufmann, 2002. Available as IfI-Report under `http://www.informatik.uni-leipzig.de/~brewka`.

G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109: 297–356, 1999.

F. Buccafurri, N. Leone, and P. Rullo. Stable models and their computation for logic programming with inheritance and true negation. *Journal of Logic Programming*, 27(1):5–43, 1996.

F. Buccafurri, N. Leone, and P. Rullo. Disjunctive ordered logic: Semantics and expressiveness. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 418–429. Morgan Kaufmann, 1998.

F. Buccafurri, N. Leone, and P. Rullo. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):854–860, Sep/Oct 2000.

J. Chomicki and J. Lobo. Monitors for history-based policies. In *Proceedings of the Intl. Workshop on Policies for Distributed Systems and Networks*, pages 57–72. Springer LNCS 1995, 2001.

J. Chomicki, J. Lobo, and S. Naqvi. A logic programming approach to conflict resolution in policy management. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 7th International Conference, KR-00*, pages 121–132. Morgan Kaufmann, 2000.

J. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129 – 187, 2003.

Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in nonmonotonic logic programs. In *Proceedings of the Fourth European Conference on Planning*, pages 169–181, 1997. URL `citeseer.nj.nec.com/dimopoulos97encoding.html`.

T. Eiter and G. Gottlob. Propositional circumscription and extended closed-world reasoning are $\Pi_2^P$-complete. *Theoretical Computer Science*, 114:231–245, 1993.

T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR system dlv: Progress report, comparisons and benchmarks. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 6th International Conference, KR-98*. Morgan Kaufmann, 1998.

H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies*, 1993.

E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1):21–70, 1992.

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

M. Gelfond and R. Son. Reasoning with prioritized defaults. In J. Dix, L.M. Pereira, and T. Przymusinski, editors, *Logic Programming and Knowledge Representation*, pages 164–223. Springer LNAI 1471, 1998.

B. Grosof. Diplomat: Compiling prioritized rules into ordinary logic programs for e-commerce applications (intelligent systems demonstration abstract). In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.

T. Janhunen, I. Niemelä, P. Simons, and J.-H. 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.

U. Junker. Preference-based search and multi-criteria optimization. In *Proceedings of the 18th National Conference in Artificial Intelligence, AAAI-02*, pages 34–40. AAAI Press, 2002.

E. Laenens and D. Vermeir. A fixpoint semantics for ordered logic. *Journal of Logic and Computation*, 1: 159–185, 1990.

N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. Technical report cs.AI/0211004, arXiv.org, November 2002.

V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence Journal*, 138(1-2):39–54, 2002.

V.W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K.R. Apt, V.W. Marek, M. Truszczyński, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999. cs.LO/9809032.

I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.

I. Niemelä and P. Simons. Efficient implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th Intl. Conference on Logic Programming and Nonmonotonic Reasoning*. Springer Verlag, 1997.

I. Niemelä and P. 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. ISBN 0792372247.

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.

C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123(1-2):185–222, 2000. URL `citeseer.nj.nec.com/sakama00prioritized.html`.

T. Schaub and K. Wang. A comparative study of logic programs with preference. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI-01*, 2001.

T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference in Artificial Intelligence (IJCAI-95)*, pages 631–639. Morgan Kaufmann, 1995.

T. Soininen. *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks.* PhD thesis, Helsinki University of Technology, Finland, 2000.

T. Syrjänen. Optimizing configurations. In *Proceedings of the ECAI Workshop W02 on Configuration*, pages 85 – 90, Berlin, Germany, August 2000.