Towards Logic Programs with Ordered and Unordered Disjunction*

Philipp Kärger[1], Nuno Lopes[2], Daniel Olmedilla[1], and Axel Polleres[2]

[1] L3S Research Center & Leibniz University of Hannover, Germany
[2] DERI Galway, National University of Ireland

Abstract. Logic Programming paradigms that allow for expressing preferences have drawn a lot of research interest over the last years. Among them, the principle of ordered disjunction was developed to express totally ordered preferences for alternatives in rule heads. In this paper we introduce an extension of this approach called Disjunctive Logic Programs with Ordered Disjunction (DLPOD) that combines ordered disjunction with common disjunction in rule heads. By this extension, we enhance the preference notions expressible with totally ordered disjunctions to *partially* ordered preferences. Furthermore, we show that computing optimal stable models for DLPODs still stays in Σ_2^p for head-cycle free programs and establish Σ_3^p upper bounds for the general case.

1 Introduction

Expressing preferences in logic programs has been a research issue in the community for quite some time now. One can distinguish two directions: preferences between rules of a logic program and preferences among literals. In both cases, typically the semantics of the approaches require a total order preference relation to be imposed in the preference expressions. But requiring total order preferences is a restriction that does not fit the world of subjective expressions: total order preferences do not allow for cases where for several options it is not known which one is preferred. But such cases, called indifferences, are common, be it due to incomplete information about the world or due to the lack of decision of a user between options.

In this paper we present an approach for preferences in logic programming that allows to specify *partially* ordered preferences among literals. We achieve this by combining two things which were handled separately until now: first, the usual disjunction common in disjunctive logic programs (DLP) [1–3]; second, the preference approach of Brewka et al. [4] called Logic Programming with Ordered Disjunction (LPOD). LPOD is an extension to logic programming that introduces a special disjunction denoted by the operator $\times$ that exploits the order of literals in a disjunction in order to express preferences among these literals. We argue that allowing either ordered or unordered disjunctions alone in the head of a

* This work has been supported by Science Foundation Ireland under the Lion project (SFI/02/CE1/I131) and by the European FP6 project inContext (IST-034718)

program's rules is not sufficient whenever it comes to statements of indifference in the preferences. Typically, one may be indifferent between some options but still prefer some others rather than defining a total order between all the options. In our approach, we propose to use the semantics of the ordered disjunction to express preferences and the disjunction to express indifferences. For example, the preference concerning the activities for a night may then look as follows (inspired by the example given in [4]):

$$pub \times (cinema \vee tv).$$

The intuition behind this expression is that *pub* is the most preferred option and, in case *pub* is not possible, both *cinema* and *tv* are equally preferred.

The remainder of the paper is structured as follows. In Section 2 we recall definitions about DLP and LPODs used later in the paper. In Section 3, we detail our new language including syntax and semantics definition. An encoding of partial order preferences in DLPODs is given in Section 4. Section 5 provides an implementation of our approach and in Section 6 general complexity results for computing optimal answer sets of a DLPOD are given. We want to point out that this section marks preliminary results in the sense that we have not yet nailed down exact complexity bounds for the newly defined language and that some proofs are admittedly sketchy due to space restrictions.

2 Preliminaries

The stable model semantics extends the typical least model semantics for logic programs (where all rules are definite Horn clauses) to so-called normal logic programs, i.e. programms allowing negation as failure in rule bodies. Logic programming under the answer set semantics, often referred to as "Answer Set Programming", further extends the stable model semantics by features such as various forms of disjunction. In the following, we review the definitions of two such forms of disjunction, which we will refer to later in the paper. Namely, we will introduce Disjunctive Logic Programs (DLPs) and Logic Programs with Ordered Disjunction (LPOD).

In this paper we will mostly restrict our elaborations and examples to propositional programs. As usual in answer set programming rules containing variables—also called rule schemata—are considered as representations of their instantiations where variables are replaced by the constants occurring in the program.

2.1 Disjunctive Logic Programming

Given disjoint sets of predicate, constant and variable symbols, σ^{pred}, σ^{con} and σ^{var} respectively, an atom can be defined as $p(t_1, \ldots, t_n)$ where $p \in \sigma^{pred}$, $t_1, \ldots, t_n \in \sigma^{con} \cup \sigma^{var}$ and n is called the arity of p. Atoms such that $n = 0$ are called *propositional*. A literal is an atom a or its negation $\neg a$ ('$\neg$' represents classical negation).

Definition 1 (DLP). *A disjunctive logic program (DLP) P is defined as a set of rules r of the form* $h_1 \vee \ldots \vee h_l \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n$ *where each h_i (b_j) is a literal and* not *represents negation as failure. We further define $Head(r) = \{h_1, \ldots, h_l\}$, $Body^+(r) = \{b_1, \ldots, b_m\}$, $Body^-(r) = \{b_{m+1}, \ldots, b_n\}$, and $Lit(P)$ as the set of all literals occurring in P.*

Variables present in a program P are assumed to be a shorthand notation representing each element of the Herbrand Universe of program P, HU_P, which corresponds to the set of all constants $c \in \sigma^{con}$ present in P. The semantics of DLPs is defined as usual by its disjunctive stable models, or answer sets, i.e., a set of literals S is an answer set of P if and only if it is a minimal Herbrand model of the Gelfond-Lifschitz reduct P^S, see [5, 1] for details.

Head-Cycle Free Logic Programs [6] are a special kind of disjunctive logic programs which will be of interest later in the paper. They are defined based on the notion of a program's dependency graph:

Definition 2 (Dependency graph). *The dependency graph of a Logic Program P is defined as a directed graph where every literal that occurs in P is represented as node l and there is an edge from l' to l if there is a rule in P such that $l \in Head(r)$ and $l' \in Body^+(r)$.*

Definition 3 (Head-Cycle Free). *P is head-cycle free if its dependency graph does not contain directed cycles that go through two literals occurring in the same rule head.*

2.2 Logic Programming with Ordered Disjunction

In [4], Brewka et al. describe so-called Logic Programs with Ordered Disjunction (LPOD) for expressing preferences in logic programming based on a special kind of disjunction called *ordered disjunction* and denoted by $\times$. It expresses a disjunction while at the same time building up a preference order between the single disjuncts. This ordered disjunction is—similarly to DLP—only allowed to appear in a rule's head. A typical example rule is "$pub \times cinema \times tv$." stating that *pub* is preferred to be true. If for some reason *pub* can not hold, *cinema* would be the second option, and so on. Due to space restrictions we ommit a formal presentation of LPODs and their semantics and refer the reader to [4].

2.3 Other Related Work

There is a lot of work about modeling and exploiting preferences in logic programs, we refer the reader to [7, 8] for a complete overview. To the best of our knowledge, none of the existing approaches to preference handling in logic programs allow for *partial* order preferences expressions. For the sake of completeness, we want to mention the work presented in [9]. There, LPODs are used as a basis for the policy language PPDL describing the behaviour of a network node and allowing for preference definitions between possible actions a node can perform. This approach models partial order preferences by assigning levels to elemets of distinct branches

of the partial order. However, this leveling approach does not work with all partial order preferences. For instance, the one described in our Example 5 later in the paper: there is no unique level assignment that keeps B and D as well as C and D incomparable but the semantics of partial orders defines both pairs as incomparable.

3 DLPOD—Disjunctive Logic Programs with Ordered Disjunction

In this section we will detail our approach to combine ordered and unordered disjunctions. In a few words: we allow both, ordered disjunctions indicated by the operator $\times$ and normal disjunctions indicated by the operator $\vee$ in a rule's head. Based on this we can extend the example given in [4] and define the rule

$$pub \times (cinema \vee tv) \leftarrow not\ sunny.$$

Here we allow an indifference between the two options *cinema* and *tv*. Intuitively, in case the body of the rule is true, a user prefers *pub* to be true, that is, to be contained in the answer set. If *pub* can not be satisfied (e.g., another rule remedies the possibility of visiting a pub), it is considered equal if either of the options *cinema* and *tv* are true. In the following we will first provide a detailed definition of how such rules look like and second, we define their exact semantics.

3.1 Syntax

Our syntax simply extends Logic Programs with Ordered Disjunction from [4] with the common disjunction '$\vee$' used in Disjunctive Logic Programming.

Definition 4 (Ordered Disjunctive Term). *An ordered disjunctive term is a (possibly nested) term of literals $C_1, \ldots, C_n$ connected by $\vee$ or $\times$. We define such terms recursively as follows.*
- *Any literal L is an Ordered Disjunctive Term.*
- *If t_1 and t_2 are Ordered Disjunctive Terms, then $(t_1 \times t_2)$ and $(t_1 \vee t_2)$ are Ordered Disjunctive Terms as well.*

We define a DLPOD as an extended logic program with an Ordered Disjunctive Term in the head:

Definition 5 (Disj. Log. Program with Ordered Disjunction). *A Disjunctive Logic Program with Ordered Disjunction (DLPOD) P is a set of rules of the form $r = Head_r \leftarrow Body_r.$ where $Body_r = B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_k$ such that all B_i $(1 \leq i \leq k)$ are literals and $Head_r$ is an Ordered Disjunctive Term. We further define $Body^+(r) = \{B_1, \ldots, B_m\}$, $Body^-(r) = \{B_{m+1}, \ldots, B_k\}$.*

In the following we define the semantics of a DLPOD by first introducing answer sets of a DLPOD and subsequently defining a preference relation among those answer sets.

3.2 Answer Sets of a DLPOD

The definition of the answer sets of a DLPOD is based on an extended notion of split programs as they are introduced in [4]. For defining split programs of a DLPOD we first define what an *Ordered Disjunctive Normal Form (ODNF)* of a rule is. Then, we show how to transform each rule's head into this normal form. Based on rules given in this normal form and on the definition of the *option* of such a rule, we can define the split programs of a DLPOD.

Definition 6 (Ordered Disjunctive Normal Form (ODNF)). *The Ordered Disjunctive Normal Form of an Ordered Disjunctive Term is*

$$\bigvee_{i=1}^{n} \bigtimes_{j=1}^{m_i} C_{i,j} = (C_{1,1} \times \ldots \times C_{1,m_1}) \vee \ldots \vee (C_{n,1} \times \ldots \times C_{n,m_n})$$

We call $(C_{i,1} \times \ldots \times C_{i,k_i})$ *the i-th* Ordered Disjunct *of the ODNF. We say that a rule* r *is in ODNF if* $Head(r)$ *is in ODNF.*

We treat arbitrarily nested DLPOD rules as shorthand for DLPOD rules in ODNF. I.e., given an ordered disjunctive term S and subterms a, b and c of S the following rewriting rules can be used to expand S to ODNF:

$$a \times (b \vee c) \Rightarrow (a \times b) \vee (a \times c) \tag{1}$$

$$(a \vee b) \times c \Rightarrow (a \times c) \vee (b \times c) \tag{2}$$

$$(a \times b) \times c \Rightarrow a \times b \times c \tag{3}$$

$$a \times (b \times c) \Rightarrow a \times b \times c \tag{4}$$

Example 1. By exhaustive application of these rules, we can transform any rule in a program into ODNF. For instance

$$pub \times (cinema \vee tv) \leftarrow not\ sunny.$$

yields the following rule in ODNF:

$$(pub \times cinema) \vee (pub \times tv) \leftarrow not\ sunny.$$

◇

Using the rewriting rules (1)–(4), hereafter we will define the semantics of a DLPOD P in terms of rules in ODNF only. We begin with the definition of the split programs of P which—intuitively—denote combinations of all *options* of each rule:

Definition 7 (Option of a rule). *Let* r *be a DLPOD rule in ODNF:*

$$\bigvee_{i=1}^{n} \bigtimes_{j=1}^{m_i} C_{i,j} \leftarrow body.$$

where m_i *is the number of literals in the* i*-th Ordered Disjunct of* r*. An option of* r *is any rule of the form* $(j_i \leq m_i)$*:*

$$
\begin{aligned}
C_{1,j_1} \vee C_{2,j_2} \vee \ldots \vee C_{n,j_n} \leftarrow \; & body, \\
& not\ C_{1,1}, not\ C_{1,2}, \ldots, not\ C_{1,j_1-1}, \\
& not\ C_{2,1}, not\ C_{2,2}, \ldots, not\ C_{2,j_2-1}, \\
& \ldots \\
& not\ C_{n,1}, not\ C_{i,2}, \ldots, not\ C_{n,j_n-1}.
\end{aligned}
$$

Example 2. The ODNF rule $(pub \times cinema) \vee (pub \times tv) \leftarrow not\ sunny.$ has the following four options (for example purposes repeated atoms are not removed):

$$
\begin{aligned}
pub \vee pub &\leftarrow not\ sunny. \\
pub \vee tv &\leftarrow not\ sunny, not\ pub. \\
cinema \vee pub &\leftarrow not\ sunny, not\ pub. \\
cinema \vee tv &\leftarrow not\ sunny, not\ pub, not\ pub.
\end{aligned}
$$

◇

Definition 8 (Split program of a DLPOD). *A split program P' of a DLPOD P is obtained by replacing each rule in P by one of its options.*

It is important to note that—in contrast to [4]—the split programs of DLPODs are *disjunctive* logic programs.

Example 3. Given the following DLPOD P:

$$
\begin{aligned}
pub \times (cinema \vee tv) &\leftarrow not\ sunny. \\
beach \vee hiking &\leftarrow sunny.
\end{aligned}
$$

We obtain the following four split programs:

1. $pub \leftarrow not\ sunny.$
   $beach \vee hiking \leftarrow sunny.$
2. $pub \vee tv \leftarrow not\ sunny, not\ pub.$
   $beach \vee hiking \leftarrow sunny.$
3. $cinema \vee pub \leftarrow not\ sunny, not\ pub.$
   $beach \vee hiking \leftarrow sunny.$
4. $cinema \vee tv \leftarrow not\ sunny, not\ pub.$
   $beach \vee hiking \leftarrow sunny.$

◇

Analogously to disjunctive logic programs, we define head-cycle-freeness [6] for DLPODs as follows:

Definition 9 (Dependency graph). *The dependency graph of a DLPOD P is the directed graph containing all literals in P as nodes such that there is an edge from l' to l iff there is a rule r in P such that $l \in Head(r)$ and $l' \in Body^+(r)$.*

Definition 10 (Head-Cycle Free). *A DLPOD P is head-cycle free if its dependency graph does not contain directed cycles that go through two literals occurring in two ordered disjuncts C_i and C_j $(i \neq j)$ of the same rule head.*

The following observation can be easily verified:

Proposition 1. *Split programs of head-cycle free DLPODs are head-cycle free.*

The possible optimal answer sets of a DLPOD are the answer sets of all split programs. In the following section we will explain in detail which answer set we call optimal according to the original DLPOD.

3.3 Optimal Answer Sets of a DLPOD

For the definition of the semantics of a DLPOD we still miss the notion of preferred answer sets of a DLPOD. So far, we have shown how the possible answer sets of a DLPOD are defined. In this section we will detail how to compare these possible answer sets in order to find the optimal ones (i.e., the most preferred answer sets according to the ordered and unordered disjunctions in the rules' heads). First, we define the Satisfaction Degree Vector as a measurement of how much an answer set satisfies a DLPOD rule:

Definition 11 (Satisfaction Degree Vector). *Let r be a DLPOD rule of the form*

$$r = \bigvee_{i=1}^{n} \bigtimes_{j=1}^{m_i} C_{i,j} \leftarrow A_1, ..., A_l, not\ B_1, ..., not\ B_k$$

and let S be a set of literals. The satisfaction degree vector *D of r in S is a vector of the form $D = (d_1, \ldots, d_n)$ representing degrees of satisfaction for each disjunct in r's head where each d_i is either a natural number or the constant ϵ. We define the dimensions of the Satisfaction Degree Vector as follows:*

1. *$D = (1, \ldots, 1)$ if (a) $Body_r^+ \not\subseteq S$, or (b) $Body_r^- \cap S \neq \emptyset$, or otherwise*
2. *$d_i = \epsilon$ if $C_{i,j} \notin S$ for all $1 \leq j \leq m_i$,*
3. *$d_i = min\{t | C_{i,t} \in S\}$.*

We denote the Satisfaction Degree Vector of r in S by $Deg_S(r)$.

Intuitively, in this definition, we assign to each Ordered Disjunct a penalty representing how much the answer set satisfies the disjunct. For each rule, these penalties build up a vector of degree values—one dimension for each disjunct. We choose degree ϵ for head disjuncts which do not overlap with S (cf. Condition *2*). With ϵ we denote that a particular disjunct does not tell anything about how much an answer set is preferred. Further, like in [4], we assign the best satisfaction degree (i.e., the vector $(1, \ldots, 1)$) in case a rule's body is not satisfied (cf. Conditions *1*): there is no reason to be dissatisfied if a rule does not apply for a particular answer set.

Example 4. Let us again consider the rule

$$r = (pub \times cinema) \vee (pub \times tv) \leftarrow not\ sunny.$$

Since this rule has two Ordered Disjuncts, any Satisfaction Degree Vector has two dimensions. The set of literals $\{pub\}$ as well as $\{sunny\}$ satisfies this rule to degree $(1, 1)$ (applied condition *3.* and condition *1.(b)*, respectively). The set $\{cinema\}$ satisfies r to degree $(2, \epsilon)$ and the set $\{tv\}$ satisfies r to degree $(\epsilon, 2)$. ⋄

Definition 12 (Preference acc. to a rule). *A set of literals S_1 is preferred to another S_2 according to a rule r (denoted as $S_1 \succ_r S_2$) iff $Deg_{S_1}(r) = (d_1^1, \ldots, d_n^1)$ Pareto-dominates $Deg_{S_2}(r) = (d_1^2, \ldots, d_n^2)$. That is, the following two conditions hold:*

1. $\forall i\ (\ d_i^1 \leq d_i^2\ \vee\ d_i^1 = \epsilon\ \vee\ d_i^2 = \epsilon\ )$
2. $\exists i\ d_i^1 < d_i^2\ \ (d_i^1 \neq \epsilon \wedge d_i^2 \neq \epsilon)$.

Intuitively, we require all dimensions in $Deg_{S_1}(r)$ to show a smaller or equal number than in $Deg_{S_2}(r)$ and in at least one dimension $Deg_{S_1}(r)$ has to show a strictly smaller number than $Deg_{S_2}(r)$. The constant ϵ plays the role of a placeholder which is, roughly speaking, equal to any number (cf. Condition *1*). As we will see in Section 4, this ϵ provides us with the "incomparability" needed to capture partial orders.

Now, we finally extend the preference notion to a relation comparing sets of literals according to a whole DLPOD:

Definition 13 (Preference acc. to a program). *A set of literals S_1 is preferred to another S_2 according to a set of rules $R = \{r_1, \ldots, r_n\}$ (denoted as $S_1 \succ S_2$) iff $\exists i(S_1 \succ_{r_i} S_2) \wedge \neg\exists j(S_2 \succ_{r_j} S_1)$.*

The conditions in both definitions follow the fair principle of Pareto optimality: an object is preferred if it is better or equal to another in all attributes (in our case in all Ordered Disjuncts or in all rules, respectively) and strictly better in at least one attribute. Finally, we provide the Definition of a preferred answer set of a program P:

Definition 14 (Preferred Answer Set). *Given a DLPOD P, one of its split programs P', and an answer set S of P'. S is called a preferred answer set (of P) if there is no answer set S' of P' for which $S' \succ S$ holds.*

4 Encoding Partial Order Preferences into DLPODs

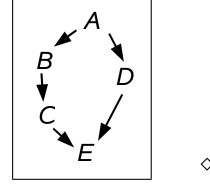
As hinted already in the introduction part, DLPOD-programs extend the approach of preferences in logic programming towards *partial* order preference relations. In this section we detail how to actually model partial order preference relations with Ordered Disjunctive Terms. For this, we specify a transformation of a partial order of literals into a Disjunctive Normal Form yielding a partial order preference statement in a rule's head.

Definition 15 (Transformation of a Partial Order). *Given a Partial Order $<$ over a set of literals S and its corresponding covering relation $<_*$ (that is, $<_*$ contains the transitive reflexive reduction of $<$), the transformation P of $<$ into an Ordered Disjunctive Term is defined as: $P(<, S) = \bigvee_{j=1}^{n}(C_1 \times \ldots \times C_{k_j})$ such that $(\forall C_i : C_i \in S) \ \wedge (\neg\exists C : C <_* C_1) \wedge (\neg\exists C : C_k <_* C) \wedge (\forall i : C_i <_* C_{i+1})$.*

Intuitively speaking, given a partial order preference relation represented by its Hasse-diagram [10], for each possible path from an element with no incoming edges to an element with no outgoing edges, we create an Ordered Disjunct $(C_1 \times \ldots \times C_k)$ where C_1 is a node with no incoming edge, C_k is a node with no outgoing edge, and there is an edge between any pair C_i, C_{i+1}.

Example 5.
Given the preference relation $<$ over the set of literals $S = \{A, B, C, D, E\}$ as depicted in the Hasse diagram on the right hand side, the transformation $P(<, S)$ yields the following Ordered Disjunctive Term:

$$(A \times B \times C \times E) \vee (A \times D \times E).$$



◇

This transformation provides us with the means for modelling partial order preferences in DLPODs: now every partial order preference expressed for literals can be formulated as the head of a rule in a DLPOD.

5 Implementation

As for a possible implementation, we extend the implementation of LPOD by Brewka et al. [11] towards DLPODs. As we shall see, this is not entirely straightforward. Concretely, in [11] the LPOD semantics is implemented on top of a standard solver for non-disjunctive logic programs based on the observation that each split program corresponds to guessing exactly one degree for each rule with ordered disjunction. Our approach and [11] basically share the following procedure to compute a preferred answer set given a program P:

1. Guess a particular satisfaction degree vector for each rule (i.e., a split program) and compute the answer sets for this guess. This is encoded in a in program called *generator* $G(P)$.
2. For each answer set S, check whether there is no split program which yields a better answer set than S. This is encoded in a program $T(P, S)$ called *tester*, which is called in an interleaved fashion for each answer set generated by $G(P)$. Whenever the tester does not find a better answer set, S is a preferred answer set.

This is analog to [11] except for the following three modifications. First, in order to generate all possible splits we need to guess a satisfaction degree *vector* per rule (instead of a single degree value). Second, we need to generate the answer sets for each split, which is—as opposed to LPODs—a *disjunctive* logic program. Third, we need to modify the *tester* program which establishes whether a better answer set can be found.

Before adapting the formal definitions of Brewka et al.'s generator and tester we need to prove two lemmata. The first Lemma states that one can replace a head symbol h in a disjunctive rule of a program P with a new symbol h' by adding some extra rules without changing the semantics of P:

Lemma 1 (Ground head atom replacement). *Let* $r = h_1 \vee \ldots \vee h_i \vee \ldots \vee h_n \leftarrow Body_r$. *be a rule in a disjunctive logic program* P *such that* h_i *is ground, and let further* $P' = P \setminus r \cup \{h_1 \vee \ldots \vee h'_i \vee \ldots \vee h_n \leftarrow Body.\ h'_i \leftarrow h_i.\ h_i \leftarrow h'_i.\}$ *such that* h'_i *does not occur in* P. *Then* S *is an answer set of* P *if and only if* $S' = S \cup \{h'_i \mid h_i \in S\}$ *is an answer set of* P'.

Similarly, we note that a part of the body of r can essentially be "outsourced" to an external rule by the following Lemma:

Lemma 2 (Body replacement). *Let $r = Head \leftarrow Body_1, Body_2.$ be a rule in a disjunctive logic program P, and let further*

$$P' = P \setminus r \cup \{Head \leftarrow b', Body_2. b' \leftarrow Body_1.\}$$

such that b' does not occur in P. Then S is an answer set of P if and only if $S' = S \cup \{b' \mid Body_1 \text{ true in } S\}$ is an answer set of P'.

Using these lemmata, we are almost ready to go ahead to define the *generator* program. For this definition we make use of the cardinality constraint notation $L\{l_1, \dots l_n\}U$ [12] in the head of a rule. Here, $l_1, \dots, l_n$ are literals and L (lower bound) and U (upper bound) are natural numbers. The intuition is that this statement holds if at least L and at most U of the literals $l_1, \dots, l_n$ are satisfied.

Definition 16 (Generator Program, adapts [11, Def. 10]). *Let r be the rule of a DLPOD of the form*

$$\begin{array}{ll} H_{1,1} \times \ldots \times H_{1,m_1} \vee & \\ \vdots & \\ H_{i,1} \times \ldots \times H_{i,m_i} \vee & \leftarrow Body_r. \\ \vdots & \\ H_{n,1} \times \ldots \times H_{n,m_n} & \end{array}$$

Then the transformation $G(r)$ is defined as the following set of rules:

(a) $\{\ 1\{c_{r,i}(1), \dots, c_{r,i}(m_i)\}1 \leftarrow\ Body_r. \mid 1 \le i \le n\}$
(b) $\cup\ \{\ h_{r,1} \vee \ldots \vee h_{r,n} \leftarrow b_{r,1}, \dots, b_{r,n}, Body_r.\}$
(c) $\cup\ \{\ h_{r,i} \leftarrow H_{i,j}, c_{r,i}(j).\ H_{i,j} \leftarrow h_{r,i}, c_{r,i}(j). \mid 1 \le i \le n, 1 \le j \le m_i\}$
(d) $\cup\ \{\ b_{r,i} \leftarrow c_{r,i}(j), not\ H_{i,1}, \dots, not\ H_{i,j-1}. \mid 1 \le i \le n, 1 \le j \le m_i\}$
(e) $\cup\ \{\ \leftarrow not\ H_{1,1}, \dots, not\ H_{1,m_1}, \dots,$
$not\ H_{i-1,1}, \dots, not\ H_{i-1,m_{i-1}},$
$not\ H_{i,1}, \dots, not\ H_{i,j-1},$
$H_{i,j}, not\ c_{r,i}(j),$
$not\ H_{i+1,1}, \dots, not\ H_{i+1,m_{i+1}}, \dots$
$not\ H_{n,1}, \dots, not\ H_{n,m_n}. \mid\ 1 \le i \le n, 1 \le j \le m_i\}$

Finally, the transformation $G(P)$ of a complete DLPOD is the union of all its transformed rules:

$$G(P) = \bigcup \{G(r) | r \in P\}.$$

Here, the newly introduced predicates $c_{r,i}$, $h_{r,i}$, $b_{r,i}$ $(1 \le i \le n)$ stand for "choice", "head", and "body" auxiliary symbols. Whereas the $c_{r,i}$ plays the role of modeling the choice of an actual degree vector, the $h_{r,i}$ and $b_{r,i}$ predicates are auxiliary symbols used according to Lemmas 1 and 2 for a particular choice. Rules (a) are guessing a particular choice option forming a split. Using this choice, rules

(b) to (e) represent the actual rules in the split program for the particular choice, by using Lemma 1 in rules (c) and Lemma 2 in rules (d). Finally, rules (e) ensure that – in case all other ordered disjuncts $k \neq i$ are false – we must choose to add $H_{i,j}$ if no better literal $H_{i,l}$ in disjunct i with $l < j$ is already in the model. [3]

Example 6. Let us consider the following rule $r = (A \times B) \vee (C \times D) \leftarrow Body$. Then, the transformation $G(r)$ looks as follows:

(a) $1\{c_{r,1}(1), c_{r,1}(2)\}1 \leftarrow Body.$
$1\{c_{r,2}(1), c_{r,2}(2)\}1 \leftarrow Body.$
(b) $h_{r,1} \vee h_{r,2} \leftarrow b_{r,1}, b_{r,2}, Body.$
(c) $h_{r,1} \leftarrow A, c_{r,1}(1).\ A \leftarrow h_{r,1}, c_{r,1}(1).$
$h_{r,1} \leftarrow B, c_{r,1}(2).\ B \leftarrow h_{r,1}, c_{r,1}(2).$
$h_{r,2} \leftarrow C, c_{r,2}(1).\ C \leftarrow h_{r,2}, c_{r,2}(1).$
$h_{r,2} \leftarrow D, c_{r,2}(2).\ D \leftarrow h_{r,2}, c_{r,2}(2).$
(d) $b_{r,1} \leftarrow c_{r,1}(1), not\ A.$
$b_{r,1} \leftarrow c_{r,1}(2), not\ A, not\ B.$
$b_{r,2} \leftarrow c_{r,2}(1), not\ C.$
$b_{r,2} \leftarrow c_{r,2}(2), not\ C, not\ D.$
(e) $\leftarrow A, not\ c_{r,1}(1), not\ B, not\ C, not\ D.$
$\leftarrow notA, B, not\ c_{r,1}(2), not\ C, not\ D.$
$\leftarrow notA, notB, C, not\ c_{r,2}(1), not\ D.$
$\leftarrow notA, notB, notC, D, not\ c_{r,2}(2).$ ◇

Proposition 2. *Let P be a DLPOD. Then (i) $G(P)$ is polynomial in the size of P and (ii) S is an answer set of $G(P)$ if and only if $S \cap Lit(P)$ is an answer set of P.*

Proof. [sketch] (i) is easy to see by looking at the rules (a) to (e). The idea for (ii) is similar to the analogous Proposition 2 in [11] where we additionally need to apply Lemma 1 and 2. Intuitively, each "guess" of the $c_{r,i}(j)$ in rules (a) yields a split program in the sense that each rule not belonging to that particular guess is "projected" away by putting $c_{r,i}(j)$ in the bodies of rules (c) and (d). By lemmas 1 and 2 now, rule (b) exactly corresponds to the guess rule in the split program corresponding to the guess modeled in (a). □

Each answer set S of $G(P)$ is subsequently tested by a tester program $T(P, S)$ for whether it is a preferred answer set.

Definition 17 (Tester Program). *Let P be a DLPOD and S be a set of literals. The tester program checking whether there is a better answer set than S is defined as follows:*

$$\begin{aligned} T(P,S) = G(P) & \\ &\cup \{O_{i,j}.\ \mid H_{i,j} \in S\} \cup \{rule(r).\ \mid r \in P\} \\ &\cup \{better(r) \leftarrow rule(r), O_{i,j}, H_{i,k}.\ \mid r \in P, 1 \leq i \leq n, 1 \leq k \leq m_i, 1 \leq j < k\} \end{aligned}$$

[3] For the interested reader, rules (a) roughly correspond to the rule in equation (8) in [11], rules (b)–(d) to rule (4) in [11], and finally rules (e) to rule (5) in [11].

$$\cup \{worse(r) \leftarrow rule(r), O_{i,k}, H_{i,j}. \mid r \in P, 1 \leq i \leq n, 1 \leq k \leq m_i, 1 \leq j < k\}$$
$$\cup \{betterRule(R) \leftarrow better(R), not\ worse(R).$$
$$worseRule(R) \leftarrow worse(R), not\ better(R).$$
$$worseSet \leftarrow worseRule(R).$$
$$betterSet \leftarrow betterRule(R), not\ worseSet.$$
$$\leftarrow not\ betterSet.$$

Intuitively, the predicate $better(r)$ fires if there is a dimension i in r's satisfaction degree vector according to S such that $T(P,S)$ found an answer set S' with a satisfaction degree vector that is better in position i. Conversely, $worse(r)$ fires if a dimension can be found where S' is worse. Note that we do not need to encode ϵ in the Tester, since the rules defining $better(r)$ and $worse(r)$, respectively, are only constructed for comparable options, i.e., pairs of literals occurring in the same disjunct of the same rule. Next, $S' \succ_r S$ (expressed by $betterRule(r)$) holds if there is a dimension where S' is better least but there is no dimension where S' is worse. Analogously, $worseRule(r)$ determines rules such that $S \succ_r S'$. By the remaining two rules and the final constraint, answer set S' only "survives", if it is better in some rule and not worse in any rule. Thus, only those answer sets $S' \succ S$ "pass", (cf. Definition 13).

Proposition 3. *Let S be an answer set of $G(P)$. If $T(P,S)$ does not have any consistent answer sets, then S is an optimal answer set of P.*

By this result, we can implement DLPOD using a standard solver for disjunctive logic programming such as GnT [13]. We further note that LPODs are just a special case of DLPODs:

Proposition 4. *LPODs are a special case of DLPODs and the preferred answer set of an LPOD computed by $G(P)$ and $T(P,S)$ correspond 1-to-1 to the preferred answer sets computed by the generator and tester presented in [11].*

Proof. [sketch] This is easy to see by the correspondence of $G(P)$ and $T(P,S)$ modulo application of lemmas 1 and 2, i.e., the answer sets of the generator and tester programs outlined in [11] only differ by the auxiliary symbols $h_{r,i}$ and $b_{r,i}$ which are introduced according to both lemmata. □

6 Complexity

In the following, we sketch some complexity results for DLPODs which mainly derive from lifting respective results from normal LPODS to the disjunctive case. At this point, we focus on establishing membership results and leave hardness proofs for future work.

Considering the complexity of finding an optimal answer set for LPODs we observe the following. Firstly, it is easy to see that determining whether an optimal answer set exists is not more difficult than determining whether "any" answer set exists, i.e. we can straightforwardly lift Theorem 1 from [11], by the Σ_2^p-completeness of disjunctive logic programs [3].

Theorem 1. *Deciding whether a DLPOD P has an optimal answer set is Σ_2^p-complete.*

The same "lifting" to the second level of the polynomial hierarchy also works for checking whether S is optimal.

Theorem 2. *Deciding whether an answer set S of a DLPOD is an optimal answer set is in Π_2^p.*

Proof. [sketch] *Membership:* analogously to [11]. □

We also conjecture hardness, but leave the proof to future work at this point. The idea would be that in variation of the proof for co-NP-hardness for the non-disjunctive LPOD case—see [11, Proof of Theorem 2]—we should be able to use, instead of a reduction of SAT, a variation of the "standard" disjunctive encoding of QSAT with two quantifier alternations into ASP, see e.g. [14].

Theorem 3. *Given a DLPOD P and a literal $l \in Lit(P)$, deciding whether there exists an optimal answer set S such that $l \in S$ is in Σ_3^p.*

Again, we conjecture hardness, but leave the in-depth investigation to future work. For the moment, let us just focus on membership, which we show by arguing that the algorithm sketched in the previous section indeed can be brought down to Σ_3^p.

Proof. Membership: First note that the algorithm from [11] can, with slight modifications, be used to solve exactly this decision problem. Namely, we need to simply add to the "outer" $G(P)$ computation the constraint "$\leftarrow$ *not* l." invalidating answer sets that do not contain l in the initial guess to G(P). Obviously, this modification yields an algorithm which is in the complexity class ${\Sigma_2^p}^{\Sigma_2^p}$. It remains to be shown that this indeed boils down to $\Sigma_3^p = \mathsf{NP}^{\Sigma_2^p}$. Here the idea is the following: As ${\Sigma_2^p}^{\Sigma_2^p} = (\mathsf{NP}^{\mathsf{NP}})^{\Sigma_2^p}$ we should be able to use the "outer" Σ_2^p oracle also to compute the "inner" NP oracle calls. In the following, we will sketch how the algorithm of Section 5 can be modified accordingly.

Note that, since G(P) is a *disjunctive* logic program, it can—following the same approach as GnT [13]—be rewritten to two *normal* logic programs: first, $Gen(G(P))$ which takes care of computing the supported models of $G(P)$ and second, $Test(G(P), M)$ which tests for each supported model M whether it is indeed a stable model. Again, the test succeeds by non-existence of an answer set for $Test(G(P), M)$.

After disambiguating symbols occurring in $T(P, M)$ and $Test(G(P), M)$ by replacing symbols within $Test(G(P), M)$ we obtain $Test'(G(P), M)$. This guarantees no "interferences" between the two test modules, which then can simply be combined into a joint tester: $T(P, M) \cup Test'(G(P), M)$. We end up in a modified algorithm for computing optimal answer sets which proceeds as follows:

- Compute an answer set of $Gen(G(P))$
- Determine whether $T(P, M) \cup Test'(G(P), M)$ has no answer set

where $Gen(\cdot)$ and $Test(\cdot,\cdot)$ are the transformations as defined in [13]. Clearly, since $Gen(G(P))$ is solvable in NP, and $T(P, M) \cup Test'(G(P), M)$ is solvable in Σ_2^p, we have shown membership of optimal answer set computation of a DLPOD in Σ_3^p. □

We note that DLPODs preserve the better computational properties when only head-cycle free programs are considered. Actually, all examples in this paper fall in this class of programs.

Theorem 4. *Given a head-cycle-free DLPOD P and a literal $l \in Lit(P)$, deciding whether there exists an optimal answer set S such that $l \in S$ is Σ_2^p-complete.*

Proof. Hardness follows immediately from hardness of this problem for non-disjunctive LPODs. As for membership, we have stated already in Proposition 1 that each split program of a head-cycle-free program is head-cycle-free again. Thus, we can observe that guessing a split and checking whether an answer set S exists such that $l \in S$ is doable in NP and likewise checking non-existence of a better answer set is in co-NP which brings the overall problem down to Σ_2^p. □

In fact, we note that using the methodology in [14] we could even obtain an algorithm encoding optimal answer set computation for head-cycle-free DLPODs into a single disjunctive logic program, instead of interleaved computations.

As next steps, we plan to experimentally compare all three possible implementations, (i) the interleaved computation from Section 5, (ii) its refinement from the proof of Theorem 3, as well as (iii) the integrated encodings for head-cycle-free programs following [14]. We note that many Σ_2^p-complete problems have more concise encodings than the metainterpreter-based encoding in [14] and plan to explore such more concise encodings for the head-cycle-free case.

7 Conclusions and Future Work

In this paper, we have presented a new approach for modelling preferences in logic programs. By extending the approach of Logic Programs with Ordered Disjunction with normal disjunction in the head of rules, we introduce partial order preference expressions for non-monotonic reasoning. We show how to transform a DLPOD into an interleaved disjunctive logic program which allows normal ASP solvers to compute preferred answer sets. Furthermore, we show that computing optimal stable models for our extension still stays in Σ_2^p for head-cycle free programs and establish Σ_3^p upper bounds for the general case.

For future work we plan to experimentally evaluate variants of the generator and tester programs provided in Section 5 with dierent ASP solvers. We remark that our considerations have so far been restricted to DLPODs in (ordered) disjunctive normal form and that the naive transformation to this normal form by applying "distributivity" rewriting rules potentially leads to exponential blowup. A generalization of the definition of the semantics to arbitrarily nested ordered disjunctive terms along with the investigation of the applicability of cheaper, structure-preserving normal form transformations is on our agenda.

In this work we focused on a Pareto-semantic based preference notion. We are aware that in [4] two other preference notions (namely cardinality-preferred and inclusion-preferred) are introduced. However, at the same time they are proven to be not general enough (see the motivation for Def. 11 in [4]). We argue that these two preferences are based on counting and hence do not reflect the qualitative nature of partial order preferences. However, we leave to future work considerations of integrating these preferences into our approach.

For basing our language extensions on solid ground, we are planning to add the hardness proofs for Theorem 2 and Theorem 3. As already stated at the end of Section 6, we plan to compare and evaluate the different implementation strategies outlined in Sections 5 and 6.

References

1. Przymusinski, T.C.: Stable Semantics for Disjunctive Programs. New Generation Computing **9** (1991) 401–424
2. Minker, J., Rajasekar, A., Lobo, J.: Foundations of Disjunctive Logic Programming. MIT Press (1992)
3. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Transactions on Database Systems **22**(3) (September 1997) 364–418
4. Brewka, G., Niemelä, I., Syrjänen, T.: Logic programs with ordered disjunction. Computational Intelligence **20** (May 2004) 335–357(23)
5. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9** (1991) 365–385
6. Ben-Eliyahu, R., Dechter, R.: Propositional Semantics for Disjunctive Logic Programs. Annals of Mathematics and Artificial Intelligence **12** (1994) 53–87
7. Delgrande, J.P., Schaub, T., Tompits, H., Wang, K.: Towards a classification of preference handling approaches in nonmonotonic reasoning. Computational Intelligence **20** (2003) 308–334
8. Niemelä, I.: Language extensions and software engineering for ASP. Technical report, European Working group on Answer Set Programming (2005)
9. Bertino, E., Mileo, A., Provetti, A.: PDL with preferences. In: Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05), Los Alamitos, CA, USA, IEEE Computer Society (2005) 213–222
10. Skiena, S.: 5.4.2 Hasse Diagrams. In: Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Addison-Wesley (1990) 163, 169–170, and 206–208
11. Brewka, G., Niemelä, I., Syrjänen, T.: Implementing ordered disjunction using answer set solvers for normal programs. In: JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence, London, UK, Springer-Verlag (2002) 444–455
12. Simons, P.: Extending the smodels system with cardinality and weight constraints. In: Logic-Based Artificial Intelligence, Kluwer Academic Publishers (2000) 491–521
13. Janhunen, T., Niemelä, I.: GnT – A Solver for Disjunctive Logic Programs. In: LPNMR 2004. (2004) 331–335
14. Eiter, T., Polleres, A.: Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. Theory and Practice of Logic Programming (TPLP) **6**(1-2) (2006) 23–60