Nested Logic Programs with Ordered Disjunction

Roberto Confalonieri and Juan Carlos Nieves

Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
C/ Jordi Girona Salgado 1-3
E - 08034 Barcelona
{confalonieri,jcnieves}@lsi.upc.edu

Abstract In this paper we define a class of nested logic programs, nested logic programs with ordered disjunction ($LPODs^+$), which allows to specify qualitative preferences by means of nested preference expressions. For doing this we extend the syntax of logic programs with ordered disjunction (LPODs) to capture more general expressions. We define the $LPODs^+$ semantics in a simple way and we extend most of the results of logic programs with ordered disjunction showing how our approach effectively is a proper generalisation of LPODs.

1 Introduction

Logic programs with ordered disjunction (LPODs) [2] are extended logic programs based on answer set semantics which combine some of the ideas underlying Qualitative Choice Logic [3] with logic programming. Indeed LPODs augment the traditional syntax of logic programs with a new *ordered disjunction* logic connective $\times$ to express preferences among literals in rules head. One distinctive characteristic of the $\times$ connective is its ability to induce an order among the answer sets of a logic program since each answer set is associated with a rule satisfaction degree which can be used to specify preference relations for answer sets selection [2]. In this ways LPODs represent a good candidate for the specification of problem solving methods targeting applications where preferences and conflicts between problem solutions are involved, such as configuration management, policy monitoring and user preference representation and reasoning [2].

The syntax of an LPOD allows the writing of rules of the kind $A_1 \times \ldots \times A_k \leftarrow B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_{m+n}$, where each of the A_i (with $1 \leq i \leq k$) and B_j's (with $1 \leq j \leq m+n$) are literals (elementary formulas), to express context-dependent preferences. Hence, they can encode simple preferences such as: *In the night I prefer going to a pub over going to a cinema over watching tv* (encoded as $pub \times cinema \times tv \leftarrow night$ [2]), but their syntax is limited when other type of preference statements have to be formulated. For example, the preference concerning the activities for a night may want to express that in case *pub* is not possible, both *cinema* and *tv* are equally preferred. This

case of preference indifference has been covered by Kärger *et al.* by means of logic programs with ordered and unordered disjunction (DLPODs) [8]. DLPODs extend LPODs combining the semantics of the ordered disjunction to express preferences and the disjunction to express indifferences. In this way their syntax allows to express preference equalities in ordered disjunction rules by means of combinations of literals connected by $\vee$. For example the indifference preference statement about *cinema* and *tv* in the night scenario can be encoded as $pub \times (cinema \vee tv) \leftarrow night$ [8].

However, they may exist more complex preference statements that neither LPODs nor DLPODs are able to express. For instance, in the night preferences above, we can be instead concerned of having both options at the same time, expressing that in case *pub* is not possible we *do* mind watching a movie in the *cinema* and not in the *tv*, *i.e.* expressions such as $pub \times (cinema \wedge \neg tv) \leftarrow night$, or that we even want to have equality and indifferences at the same time, *i.e.* expressions such as $(pub \vee bar) \times (cinema \wedge \neg tv) \leftarrow night$, or even more complex preference expression such as $(pub \wedge (expensive \times cheap)) \times bar \times (not\ pub \wedge not\ bar) \leftarrow night \vee (not\ busy \wedge afternoon)$. These examples suggest to explore for a less restricted syntax language.

For this reason, in this paper we present a more general syntax which allows the writing of nested (or non-flat) preferences expressions built by means of connectives $\{\vee, \wedge, \neg, not, \times\}$. To represent these formulas and to capture their semantics we define an extension of logic programs with ordered disjunction called *nested logic programs with ordered disjunction* ($LPODs^+$).

The syntax of $LPODs^+$ is based on a language which allows nested formulas that can provide a richer syntax at the moment of writing conditional preferences. The language we use is constructed using as a basis the propositional language of Qualitative Choice Logic [3], extended to consider negation as failure *not*, a typical connective which characterises non-monotonic reasoning in Answer Set Programming (ASP) [1].

To define the semantics of our $LPODs^+$ we consider and extend some of the results in [9] where the semantics for nested logic programs is presented. In this way we can capture the semantics of an $LPODs^+$ in a simple way and we show how when the formulas in an $LPOD^+$ are $\times$ free, the $LPODs^+$ semantics and the semantics of nested logic programs coincide. Moreover, when an $LPOD^+$ syntactically corresponds to an LPOD the two semantics coincide as well.

In the presence of nested expression in the head of preference rules, the determination of the degree of satisfaction of an answer set of an $LPOD^+$ can be sometimes more involved in our approach. For this reason we propose a recursive function to calculate the optionality of complex preference formulas in order to determine the rule satisfaction degree. The optionality function is a generalisation of the rule satisfaction degree in LPODs, as a consequence, we can directly use the preference relations in [2] for comparing the answer sets of an $LPOD^+$.

Our paper is structured as follows: after introducing the language we adopt to define our logic programs (Section 2), in Section 3 we present the syntax and the semantics of $LPODs^+$, and the optionality function for answer sets selec-

tion. Finally Section 4 provides some prelimary discussions about related works, $LPOD^+$ complexity and sketches an implementation design of an $LPOD^+$ solver. Along the paper we present a running example which exemplifies the definitions of our approach. Due to space reasons we are not able to provide an extensive comparison with related approaches and we leave proofs' results out of the paper. It is our intention to cover these missings in an extended version.

2 Language Considered

The language we will consider to define the syntax of our logic programs is based on the language of Qualitative Choice Logic (QCL) extended with negation as failure *not*.

QCL is a propositional logic for representing alternatives for problem solutions defined by Brewka *et al.* in [3] which adds to classical propositional logic [6] a new connective $\times$ called ordered disjunction.[1]

The language we consider consists with: (i) an enumerable set $\mathcal{L}$ of elements called *atoms* (denoted a, b, c, ...), (ii) *connectives* $\wedge$, $\vee$, $\times$, $\neg$, *not*, $\bot$, $\top$ where $\{\wedge, \vee, \times\}$, $\{not, \neg\}$, $\{\top,\bot\}$ are 2-place, 1-place and 0-place connectives respectively and (iii) *auxiliary symbols* "(", ")", ".". If an atom a is negated by $\neg$ is known as negative literal, and a is known as positive literal if it is not preceded by $\neg$.

Literals, $\bot$ and $\top$ are considered *elementary formulas*, while formulas (denoted A, B, C) are constructed from elementary formula using the connectives $\{\vee, \wedge, \neg, not, \times\}$ arbitrarily nested.[2]

A *theory* is a set of formulas and a *logic program* corresponds to a finite theory, *i.e.* a finite set of formula.

Based on this language we will define in Section 3 the syntax of nested logic programs with ordered disjunction. In the tradition of logic programming we write conditional expressions as the formula $B \leftarrow A$. We will consider two types of negation in our logic programs: strong negation $\neg$ and negation as failure *not*. Intuitively *not* a is true whenever there is no reason to believe a, whereas $\neg a$ requires a proof of the negated atom. In this paper we assume that $\mathcal{L}$ is a finite set and we restrict our attention to finite propositional theories, since the semantics can be extended to programs with variables by grounding. Function symbols are, however, not allowed to ensure the ground program to be finite. This is a standard procedure in ASP.

3 $LPODs^+$

This section presents the syntax of nested logic programs with ordered disjunction ($LPOD^+$) and their semantics which is characterised in terms of a recursive

[1] The original connective is denoted by $\overrightarrow{\times}$, however we write $\times$ for consistent notation with ordered disjunction used in Answer Set Programming [2,4].

[2] In the following we assume that the connectives $\wedge$, $\vee$, and *not* have stronger bindings than $\times$. We also assume that $\times$ is associative.

Table 1. Example of rules captured by $LPOD^+$ syntax

Syntax	Rule Type
$(a \wedge (b \times c)) \times (d \vee e) \leftarrow g \vee (not\ i \wedge f)$.	*Nested Ordered Disjunction rule*
$a \vee (b \times \neg c) \leftarrow d \vee (e \wedge f)$.	*Definite Nested Ordered Disjunction rule*
$a \wedge b \leftarrow p \wedge (\neg q \vee r)$.	*Definite Nested rule* [9]
$a \times (b \vee c) \leftarrow d \wedge\ not\ e$.	*Ordered Disjunctive rule* [8]
$a \times b \leftarrow c \wedge not\ d$.	*Ordered Disjunction rule* [2]
$a \vee b \leftarrow c \wedge not\ \neg e$.	*Disjunctive rule* [7]
$a \wedge not\ b \leftarrow p \wedge not\ (\neg q \vee r)$.	*Nested rule* [9]

reduction. We also define a recursive function for calculating the rule satisfaction degree of an answer set of an $LPOD^+$ for comparing answer sets.

3.1 Syntax

Earlier in the paper we have pointed out how the syntax of existent logic programming approaches able to express qualitative preferences is usually quite restricted and for this reason we define a more expressive syntax.

Let $\mathcal{L}$ be a set of atoms, then a *nested ordered disjunction rule* (rule for short) is an expression of the form $H \leftarrow B.$, where H is either an elementary formula or a $\{\vee, \wedge, \neg, not, \times\}$ formula (known as the *head*) and B is either an elementary formula or a $\{\vee, \wedge, \neg, not\}$ formula (known as the *body*) built using the atoms in $\mathcal{L}$. Some particular cases are *facts*, of the form $H \leftarrow \top$ (written as H) and *constraints*, $\bot \leftarrow B$ (written as $\leftarrow B.$). If no occurrences of *not* appear in a rule, the rule is a *definite nested ordered disjunction rule* (similarly a definite nested ordered disjunction formula). If no occurrences of $\times$ appear in a rule, the rule is a *nested rule* (similarly a nested formula). If no occurrences of $\times$ and *not* appear in a rule, then the rule is known as a *definite nested rule* (similarly a definite nested formula). Different formulas combinations can lead to different rules (some of them already defined in the literature) as shown in Table 1.

A *nested logic program with ordered disjunction* ($LPOD^+$) is a finite set of nested ordered disjunction rules and/or constraints and/or facts. If the program does not contain *not* the program is called a *definite* $LPOD^+$. The set of all the atoms which appear in an $LPOD^+$ P is denoted by $\mathcal{L}_P$.

In our logic programs we will manage the strong negation $\neg$ as it is done in ASP [1]. Basically, each negative literal $\neg a$ is replaced by a new atom symbol a' which does not appear in $\mathcal{L}_P$ and we add the constraint $\leftarrow a, a'$ to the program. For managing the constraints in our logic programs, we will replace each rule of the form $\leftarrow B$ by a new rule of the from $f \leftarrow B,\ not\ f$ such that f is a new atom symbol which does not appear in $\mathcal{L}_P$.

The use of $\{\vee, \wedge, \neg, not, \times\}$ formulas in the head of nested ordered disjunction rules, rather than elementary formulas or disjunctive formulas only (as in [2,8]), provides a richer expressiveness in specifying qualitative preferences in our logic programs. Using $\vee$, and $\wedge$ we can express for instance that certain options are equally preferred or that certain combinations are preferred over other

combinations. The following program exemplifies some preference statements we can express by means of an $LPOD^+$.

Example 1. Let P be an $LPOD^+$ representing the user preferences of the form:
$r_1 = italian \times peruvian \times (not\ italian \wedge not\ peruvian)$.
$r_2 = (pub \vee bar) \times (cinema \wedge \neg tv) \leftarrow night.$
$r_3 = night.$
Briefly, the intuitive reading of each of the rules of the program P is the following: r_1 expresses that we generally prefer to have *italian* over *peruvian* food and we prefer to have one of the options over not having any of them; r_2 tells that in the *night* we prefer to go to a *pub* or a *bar*, but if it not possible then we wish to see a movie in the *cinema* and not in the *tv* and r_3 just tells that we imperatively want to do something in the *night* time.

In the next section we define the semantics for inferring the answer set of an $LPOD^+$.

3.2 Semantics

Keeping in mind that the definition of answer sets semantics [7] consists of two parts (a syntactic reduction and a semantics for definite programs), the extension of this definition to $LPOD^+$ follows the same strategy. Thus, to define the semantics of our $LPODs^+$ we consider and extend some of the results in [9] where a semantics for nested logic programs is presented. The first definitions are simply reported as they represent some basic results we reuse for achieving our scope. Instead, Definition 4 and 5 extend the original ones for nested programs to consider the $\times$ connective which characterizes our approach.

Definition 1. *[9] Let M be a set of atoms. M satisfies a definite nested formula A (denoted by $M \models A$), recursively as follows:*

- *for elementary A, $M \models A$ if $A \in M \vee A = \top$*
- *$M \models A \wedge B$ if $M \models A \wedge M \models B$*
- *$M \models A \vee B$ if $M \models A \vee M \models B$*

Definition 2. *[9] Let P be a definite nested logic program. A set of atoms M is closed under P if, $\forall r \in P$, $M \models H$ whenever $M \models B$.*

Definition 3. *[9] Let M be a set of atoms and P a definite nested logic program. M is called an answer set for P if M is minimal among the sets of atoms closed under P.*

Definition 4. *The reduct of a nested ordered disjunction formula with respect a set of atoms M is defined recursively as follows:*

- *for elementary A, $A^M = A$*
- *$(A \wedge B)^M = A^M \wedge B^M$*
- *$(A \vee B)^M = A^M \vee B^M$*

– $(not\ A)^M \begin{cases} \bot, & \text{if } M \models A^M \\ \top, & \text{otherwise} \end{cases}$

– $(A \times B)^M \begin{cases} A^M, & \text{if } M \models A^M \\ B^M, & \text{if } M \not\models A^M \wedge M \models B^M \\ \bot, & \text{otherwise} \end{cases}$

Definition 5. *The reduct $P^M_{\times^+}$ of a nested logic program with ordered disjunction with respect to a set of atoms M is defined recursively as follows:*

– $(H \leftarrow B)^M = H^M \leftarrow B^M$
– $P^M_{\times^+} = \{(H \leftarrow B)^M \mid H \leftarrow B \in P\}$

Please notice that $P^M_{\times^+}$ is a definite nested logic program, *i.e.* $\times$ and *not* free. Hence, the following definition follows in a straightforward way from the answer set definition for definite nested logic programs.

Definition 6. *Let P be an $LPOD^+$ and M a set of atoms. M is answer set of P if it is an answer set of $P^M_{\times^+}$.*

Example 2. Let us consider the $LPOD^+$ P in Example 1[3] and the set of atoms $M = \{b, d, g\}$. Then $P^{\{b,d,g\}}_{\times^+}$ can be obtained in three recursive steps applying Definition 4 and 5:

(1)	(2)	(3)
$(a \times b \times (not\ a \wedge not\ b))^{\{b,d,g\}}.$	$b^{\{b,d,g\}}.$	$b.$
$(c \vee d) \times (e \wedge f') \leftarrow g)^{\{b,d,g\}}.$	$(c \vee d)^{\{b,d,g\}} \leftarrow g^{\{b,d,g\}}.$	$c \vee d \leftarrow g.$
$(g)^{\{b,d,g\}}.$	$g.$	$g.$
$(\leftarrow f \wedge f'.)^{\{b,d,g\}}.$	$\leftarrow f^{\{b,d,g\}} \wedge f'^{\{b,d,g\}}.$	$\leftarrow f \wedge f'.$

Clearly, M is an answer set of $P^{\{b,d,g\}}_{\times^+}$ and so M is an answer set of P. Similarly, it can be proved that the sets of atoms $\{a, c, g\}$, $\{a, d, g\}$, $\{a, e, f', g\}$, $\{b, c, g\}$, $\{b, e, f', g\}$, $\{c, g\}$, $\{d, g\}$ and $\{e, f', g\}$ are valid answer sets of P as well.

It can be noticed that there is an interesting property of our $LPOD^+$ semantics. Besides the fact that $LPOD^+$ has a richer syntax they show to have a richer semantics as well. When all the rules of an $LPOD^+$ P are ordered disjunction rules (according to the syntax of LPODs in [2]), the $LPOD^+$ semantics and the LPODs semantics in fact coincide.

Proposition 1. *Let P be an $LPOD^+$ such that $\forall r \in P$, $r = A_1 \times \ldots \times A_k \leftarrow B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_{m+n}$ each A_i (with $1 \leq i \leq k$) and B_j'a (with $1 \leq j \leq m+n$) are literals (or elementary formulas), and M be a set of atoms. M is answer set of $P^M_{\times^+}$ if and only if M is answer set of $P^M_{\times}$.*[4]

[3] Due to representation issues we have changed the signature of the program from $\{italian, peruvian, pub, bar, cinema, tv, tv'night\}$ to $\{a, b, c, d, e, f, f'g\}$ respectively.

[4] The $P^M_{\times}$ reduction is defined as $\bigcup_{r \in P} r^M_{\times}$, where $r^M_{\times} := \{A_i \leftarrow B_1, \ldots, B_m | A_i \in M \text{ and } M \cap (\{A_1, \ldots, A_{i-1}\} \cup \{B_{m+1}, \ldots, B_{m+n}\}) = \emptyset\}$ (see [2] for details).

Another interesting property is that when the formulas in our $LPODs^+$ are formulas without the $\times$ connective, then the $LPODs^+$ semantics and the semantics for nested logic programs defined in [9] coincide.

Proposition 2. *Let P be an $LPOD^+$ such that $\forall r \in P$, $r = H \leftarrow B$, H and B are well-formed formulas $\times$ free, and M be a set of atoms. M is answer set of $P^M_{\times^+}$ if and only if M is answer set of Π^M.*[5]

3.3 Answer Sets Selection

At the beginning of the paper we have pointed out how an $LPOD^+$ is a specification to express preferences about certain conditions inducing a preference order among its answer sets. Thus, the key question now is how such ordering can be achieved. In the simplest case where each rule of an $LPOD^+$ corresponds to an ordered disjunction rule, a satisfaction degree k (with $1 \leq i \leq k$) can be associated to an answer set M *w.r.t.* a rule ($deg_M(r)$) which corresponds to the position of the best satisfied literal [2].

However, when we consider a nested ordered disjunction rule $r = H \times B$ where H and B can be formulas built from $\{\vee, \wedge, \neg, not, \times\}$ and $\{\vee, \wedge, \neg, not\}$ respectively, the assignation of a satisfaction degree is not so trivial. Let us assume in fact that M is an answer set of an $LPOD^+$ P. How is possible to determine the satisfaction degree of M *w.r.t.* each rule r in P? This degree may depend on how many options H admits. For this reason we first define the *optionality* of an answer set M *w.r.t.* a nested ordered disjunction formula H as follows:

Definition 7. *Let H be a $\{\vee, \wedge, \neg, not, \times\}$ formula, and M be an answer set of an $LPOD^+$ P. Then the optionality of M w.r.t. H, denoted by $opt_M(H)$ is recursively defined as follows:*

$$opt_M(H) \begin{cases} 1, \text{ if } H = A \text{ and } A \text{ is an elementary formula} \\ k \begin{cases} \text{if } H = (not\ P) \text{ then } k = opt_M(P) \\ \text{if } H = (P \wedge Q) \text{ then } k = max\{opt_M(P), opt_M(Q)\} \\ \text{if } H = (P \vee Q) \text{ then } k = max\{opt_M(P), opt_M(Q)\} \\ \text{if } H = (P \times Q) \text{ then } \begin{cases} k = opt_M(P), \text{ if } M \models P^M \\ k = opt_M(P) + opt_M(Q), \text{ otherwise} \end{cases} \end{cases} \end{cases}$$

Based on this function, the satisfaction degree of an answer set M *w.r.t.* a nested ordered disjunction rule $r = H \leftarrow B$ can be defined as:

Definition 8. *Let M be an answer set of an $LPOD^+$ P. Then the satisfaction degree M w.r.t. a rule $r = H \leftarrow B$, denoted by $deg^+_M(r)$ is:*

$$deg^+_M(r) \begin{cases} 1, & \text{if } M \not\models B^M \\ opt_M(H), & \text{otherwise} \end{cases}$$

[5] The Π^M reduction is defined as the $P^M_{\times^+}$ without the $\times$ case (see [9] for details).

The degrees can be viewed as penalties, in fact a higher degree expresses a lesser satisfaction. Thus, 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 [2,3].

We can observe that there is a direct property *w.r.t.* the optionality function we defined for the answer sets of an $LPOD^+$. The optionality function clearly generalises the satisfaction degree of answer sets of an LPOD.

Proposition 3. *Let P be an $LPOD^+$ such that $\forall r \in P$, $r = A_1 \times \ldots \times A_k \leftarrow B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_{m+n}$ each A_i (with $1 \leq i \leq k$) and B_j's (with $1 \leq j \leq m+n$) are literals (or elementary formulas), and M be an answer set of P. Then $deg^+_M(r) = deg_M(r)$.*[6]

Therefore our approach also generalises in a consistent way the preference relation between answer sets of LPODs.

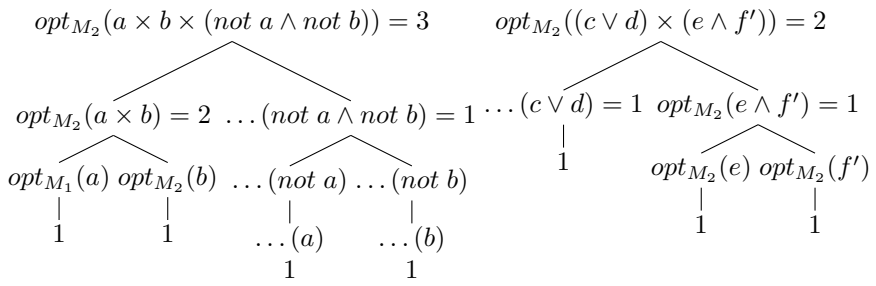
Definition 9. [2] *M_1 is preferred to M_2 (denoted by $M_1 > M_2$) iff $\exists\ r \in P$ such that $deg^+_{M_1}(r) < deg^+_{M_2}(r)$, and $\nexists r' \in P$ such that $deg^+_{M_2}(r') < deg^+_{M_1}(r')$.*

Example 3. Let P be the $LPOD^+$ in Example 1 and $M_1 = \{a, c, g\}$, $M_2 = \{e, f', g\}$ be two of the answer sets of P in Example 2. We want to show that $M_1 > M_2$. For doing this, we first have to calculate the satisfaction degrees of M_1 and M_2 *w.r.t.* all the rules in P. Applying Definition 7 and 8 we can see how the degrees of M_1 *w.r.t.* r_1 and r_2 correspond to $deg^+_{M_1}(r_1) = 1$ and $deg^+_{M_1}(r_2) = 1$ respectively since:

$$\begin{array}{c} opt_{M_1}(a \times b \times (not\ a \wedge not\ b)) = 1 \\ | \\ opt_{M_1}(a \times b) = 1 \\ | \\ opt_{M_1}(a) \\ | \\ 1 \end{array} \qquad \begin{array}{c} opt_{M_1}((c \vee d) \times (e \wedge f')) = 1 \\ | \\ opt_{M_1}(c \vee b) = 1 \\ | \\ 1 \end{array}$$

The case of M_2 looks more interesting as it shows how determining the degree of satisfaction can be sometimes more involved. As M_2 satisfies only the third condition in the head of rule r_1, its optionality *w.r.t.* this rule has to take into account that the first two conditions are not satisfied, *i.e.* $deg^+_{M_2}(r_1) = 2 + opt_{M_1}(not\ a \wedge not\ b)$. As shown below the degrees of M_2 *w.r.t.* r_1 and r_2 correspond to $deg^+_{M_2}(r_1) = 3$ and $deg^+_{M_2}(r_2) = 2$ respectively since:

[6] We refer to [2] for the definition of $deg_M(r)$.



The simplest case of r_3 is not represented as r_3 is clearly satisfied by degree 1 by both answer sets. Once the degrees are calculated the preference relation can be used to compare M_1 and M_2. It is easy to see how M_1 is preferred to M_2 since M_1 satisfies the rules of the program in a better way ($M_2 \not\succ M_1$ from Definition 9 as well).

4 Conclusions, Discussions and Future Research

The main scope of this paper has been to define the syntax and the semantics for nested logic programs with ordered disjunction. The syntax of an $LPOD^+$ is based on well-formed formulas built from connectives $\{\vee, \wedge, \neg, not\}$ plus an ordered disjunction connective $\times$ which was first introduced in QCL and then used in logic programs with ordered disjunction. As a result, rules in $LPODs^+$ are more general than rules which can be specified in related approaches of qualitative context-dependent preferences between literals such as LPODs [2] and DLPODs [8]. In fact compared to LPODs and DLPODs, the $LPOD^+$ syntax allows the specification of more complex formulas built by means of $\{\vee, \wedge, \neg, not, \times\}$ formulas in the head of the rules and $\{\wedge, \vee, \neg, not\}$ formulas in the body. In this respect we extend the syntax of LPODs and DLPODs rules where $\times$ and $\times, \vee$ literals' combinations are allowed. Among quantitative approaches to preference specification, it is worth to mention the work of Costantini *et al.* [5], where an extension of ASP is designed to model quantitative resources and enabling the specification of non-linear preferences (both in the heads and in the bodies).

We have shown how the $LPOD^+$ semantics can be defined in a lighter way (Definition 4, 5 and 6) than then LPODs semantics (which is based on split programs [2]) reusing and extending some definitions related to the semantics of nested logic programs [9]. We have also seen how when an $LPOD^+$ syntactically corresponds to an LPOD the two semantics coincide (Proposition 1) and when the formulas of an $LPODs^+$ are $\times$ free, the $LPOD^+$ semantics coincide with the semantics of nested logic programs (Proposition 2). As $LPODs^+$ allow complex formulas in their rules head, we have characterised the degree of satisfaction of an answer set *w.r.t.* a nested ordered disjunction rule (Definition 8) by means

of a recursive optionality function (Definition 7). As the optionality function generalises the satisfaction degree defined for LPODs (Proposition 3) we have been able to reuse the preference relation criteria specified in [2] for comparing the answer sets of an $LPOD^+$.

As far the complexity of reasoning is concerned, we have reasons to believe that the complexity of computing the answer sets of an $LPOD^+$ corresponds to the complexity of disjunctive logic programs. This can be informally motivated by the fact that by means of a transformation, we are currently investigating, which is able to convert nested ordered disjunction rules to $\times$ free ones (*i.e.* nested rules), we can reuse the polynomial translation for nested logic programs into disjunctive ones presented by Sarsakov *et al.* in [10]. Therefore, the implementation of an $LPOD^+$ solver can be sketched as follows: after extending the compiler for nested logic programming (**nlp**[7]) to treat the $\times$ connector, the answer sets of an $LPODs^+$ are computable by a disjunctive logic programming system such as DLV[8]. These results are preliminary and we need to explore them in a more precise way in an extended version of this paper.

Beside these extensions, as future work we are considering to extend the $LPOD^+$ formalism to be able to reason under incomplete evidence and partially inconsistent knowledge associating to each rule of an $LPOD^+$ a certainty degree following the same spirit as in our framework of logic programs with possibilistic ordered disjunction (LPPODs) we presented in [4]. The work in this paper represents in fact the first step towards an extension of the LPPODs framework. Last but not least, we aim to look for several possible applications we can target with our specification. So far, $LPODs^+$ seem prominent to model application domains such as qualitative decision making with preferences and preference queries to databases.

Acknowledgements The authors would like to thank the anonymous referees for their useful suggestions and comments. This research has been partially supported by ICT-ALIVE (FP7-215890).

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
2. Brewka, G., Niemelä, I., Syrjänen, T.: Logic Programs with Ordered Disjunction. Computational Intelligence **20**(2) (2004) 333–357
3. Brewka, G., Benferhat, S., Le Berre, D.: Qualitative Choice Logic. Artificial Intelligence **157**(1-2) (2004) 203–237
4. Confalonieri, R., Nieves, J.C., Osorio, M., Vázquez-Salceda, J.: Possibilistic Semantics for Logic Programs with Ordered Disjunction. In Link, S., Prade, H., eds.: FoIKS 2010. Volume 5956 of LNCS., Berlin,Heidelberg, Springer-Verlag (2010)

[7] `http://www.cs.uni-potsdam.de/~torsten/nlp/`
[8] `http://www.dbai.tuwien.ac.at/proj/dlv/`

5. Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. Journal of Algorithms **64**(1) (2009) 3–15
6. van Dalen, D.: Logic and Structure. Third, augmented edition edn. Springer-Verlag, Berlin (1994)
7. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9**(3/4) (1991) 365–386
8. Kärger, P., Lopes, N., Olmedilla, D., Polleres, A.: Towards Logic Programs with Ordered and Unordered Disjunction. Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP), ICLP 2008, (2008) 46–60
9. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence **25**(3-4) (1999) 369–389
10. Sarsakov, V., Schaub, T., Tompits, H., Woltran, S.: nlp: A Compiler for Nested Logic Programming. In Lifschitz, V., Niemelä, I., eds.: LPNMR04. Volume 2923 of LNCS., Springer (2004) 361–364