A Logical Characterisation of Ordered Disjunction

Pedro Cabalar*

Department of Computer Science,
Corunna University (Corunna, Spain),
cabalar@udc.es

Abstract. In this paper we consider a logical treatment for the ordered disjunction operator $\times$ introduced by Brewka, Niemelä and Syrjänen in their Logic Programs with Ordered Disjunctions (LPOD). LPODs are used to represent preferences in logic programming under the answer set semantics. Their semantics is defined by first translating the LPOD into a set of normal programs (called split programs) and then imposing a preference relation among the answer sets of these split programs. We concentrate on the first step and show how a suitable translation of the ordered disjunction as a derived operator into the logic of Here-and-There allows capturing the answer sets of the split programs in a direct way. We use this characterisation not only for providing an alternative implementation for LPODs, but also for checking several properties (under strongly equivalent transformations) of the $\times$ operator, like for instance, its distributivity with respect to conjunction or regular disjunction. We also make a comparison to an extension proposed by Kärger, Lopes, Olmedilla and Polleres, that combines $\times$ with regular disjunction.

1 Introduction

Based on the answer set (or stable model) semantics [10] for logic programs, Answer Set Programming (ASP) [17, 18] has become a successful paradigm for declarative problem solving. Typically, a logic program in ASP is used to encode some constraint-based problem in such a way that the answer sets of the program correspond to the problem solutions. In many practical scenarios, however, the set of feasible solutions is considerably large and the main problem, from the knowledge representation viewpoint, is not specifying them but selecting the most preferred ones under certain criteria instead. Although different approaches for representing preferences in ASP have been proposed (see [5] for a survey), one that has recently received much attention is the formalism of *Logic Programs with Ordered Disjunction* (LPOD) [3], probably due to its simplicity and expressiveness. This approach essentially consists in introducing a new operator '$\times$' standing for *ordered disjunction* (with its corresponding semantics

* This research was partially supported by Spanish MEC project TIN-2009-14562-C05-04 and Xunta de Galicia project INCITE08-PXIB105159PR.

in terms of answer sets), plus several ordering relations for selecting preferred models among the obtained answer sets. LPODs have been applied, for instance, in Game Theory [9], for implementing policy languages with preferences [16, 2], or in planning and argumentation scenarios [21], and they have been further investigated in [6] for studying strongly equivalent transformations and in [14] for introducing an extension called *disjunctive* LPOD (DLPOD) that combines ordered and regular disjunctions. Other ASP extensions like CR-Prolog, have also incorporated the use of ordered disjunctions [1]. The semantics of an LPOD is defined in two steps. First, the program with ordered disjunctions is translated into a set of normal programs, called *split programs*, whose answer sets become the potential solutions. In a second step, one of three possible preference relations is imposed among the answer sets of these split programs. These answer sets of the split programs can also be captured by reduct transformations, like the originally introduced in [3] or the one later proposed in [6].

In this paper we concentrate on the first step, that is, in the definition of potential answer sets for LPODs, and show that they can be directly captured by a suitable definition of the ordered disjunction connective '$\times$' as a derived operator in the logic of *Here-and-There* (**HT**) [11], so that LPODs can be seen as nothing else but regular theories inside the nonmonotonic formalism of *Equilibrium Logic* [19] (the nonmonotonic version of **HT**). Equilibrium Logic has been extensively studied in ASP, as it yields a logical characterisation for the answer set semantics, capturing concepts such as the strong equivalence of programs [15] and providing a means to generalise all previous extensions to the the most general syntax of arbitrary propositional [8] and first order [7] theories. Our logical characterisation of $\times$ allows not only an alternative method for translating LPODs into regular logic programs, but enables the study of '$\times$' as a logical connective, so we can analyse its main properties like its behaviour with respect to distributivity or nesting with other connectives.

The rest of the paper is organised as follows. The next section contains an overview of the basic definitions of Equilibrium Logic, ASP and LPODs. Section 3 introduces the characterisation of ordered disjunction and studies some of its main properties, including the correspondence to the original definition of answer sets for LPODs. Section 4 contains a comparison to DLPODs and finally, Section 5 concludes the paper.

2 Preliminaries

We recall the basic definitions of the propositional[1] logic of **HT** and Equilibrium Logic. The syntax is the same as in classical propositional logic: a *well formed formula* results from combining atoms in a finite set Σ (called the *signature*) with the usual operators $\rightarrow, \wedge, \vee, \bot$ and parentheses. We assume the standard precedence among binary operators, that is, $\wedge \prec \vee \prec \rightarrow$. The formulas $\neg F$, $\top$, $F \leftarrow G$ and $F \leftrightarrow G$ are abbreviations that respectively stand for $F \rightarrow \bot$,

[1] For simplicity sake, we will focus here on ordered disjunction inside programs without variables.

$\bot \rightarrow \bot$, $G \rightarrow F$ and $(F \rightarrow G) \wedge (F \leftarrow G)$. As usual, a *literal* is an atom p or its negation $\neg p$.

Given a set of atoms I and a formula F, we write $I \models F$ to represent classical satisfaction. The semantics of **HT** starts from defining an *interpretation* as a pair $\langle H, T\rangle$ of sets of atoms (standing for "here" and "there") where $H \subseteq T$. We say that an interpretation $\langle H, T\rangle$ *satisfies* a formula F, by abuse of notation written $\langle H, T\rangle \models F$ as in classical logic, when one of the following recursive conditions hold:

1. $\langle H, T\rangle \models p$ if $p \in H$, for any atom p.
2. $\langle H, T\rangle \not\models \bot$.
3. $\langle H, T\rangle \models F \wedge G$ if $\langle H, T\rangle \models F$ and $\langle H, T\rangle \models G$.
4. $\langle H, T\rangle \models F \vee G$ if $\langle H, T\rangle \models F$ or $\langle H, T\rangle \models G$.
5. $\langle H, T\rangle \models F \rightarrow G$ if both (i) $T \models F \rightarrow G$; and (ii) $\langle H, T\rangle \not\models F$ or $\langle H, T\rangle \models G$.

Ambiguity in the use of $\models$ is removed depending on the interpretation we use in the left hand. Thus, note that in condition (i) of line 5, we are actually referring to classical satisfaction. Some useful well-known properties relating **HT** and classical satisfaction are mentioned below.

Proposition 1. *For any interpretation $\langle H, T\rangle$ and any formula F:*

1. *$\langle H, T\rangle \models F$ implies $T \models F$.*
2. *$\langle T, T\rangle \models F$ iff $T \models F$.*
3. *$\langle H, T\rangle \models \neg F$ iff $T \models \neg F$.* □

By $\equiv_c$ we understand equivalence in classical logic. A *theory* is a set of formulas. As usual, an interpretation $\langle H, T\rangle$ is said to be a model of a theory H, also written $\langle H, T\rangle \models H$, when $\langle H, T\rangle$ satisfies all the formulas in H.

The following are some **HT** valid equivalences:

$$\neg(F \vee G) \leftrightarrow \neg F \wedge \neg G \tag{1}$$
$$\neg(F \wedge G) \leftrightarrow \neg F \vee \neg G \tag{2}$$
$$F \wedge (G \vee H) \leftrightarrow (F \wedge G) \vee (F \wedge H) \tag{3}$$
$$F \vee (G \wedge H) \leftrightarrow (F \vee G) \wedge (F \vee H) \tag{4}$$
$$F \wedge \neg F \leftrightarrow \bot \tag{5}$$
$$(F \rightarrow (G \rightarrow H)) \leftrightarrow (F \wedge G \rightarrow H) \tag{6}$$

For instance, **HT** satisfies De Morgan's laws (1), (2), and distributivity (3), (4).

Definition 1 (Equilibrium model). *An* **HT** *interpretation $\langle T, T\rangle$ is an* equilibrium model *of a theory H if $\langle T, T\rangle \models H$ and there is no $H \subset T$ such that $\langle H, T\rangle \models H$.* □

Equilibrium Logic is the (nonmonotonic) logic induced by equilibrium models. An interesting concept for nonmonotonic reasoning is the idea of strong equivalence. Two theories H_1 and H_2 are said to be *strongly equivalent*, written $H_1 \equiv_s H_2$, if $H_1 \cup \Delta$ and $H_2 \cup \Delta$ have the same equilibrium models, for any theory Δ.

Theorem 1 (From [15]). *Two theories H_1, H_2 are strongly equivalent iff they are equivalent in* **HT**. □

2.1 Logic programs

We begin introducing some preliminary notation that will be useful later. Let $\mathbf{A}$ be a (possibly empty) list of (possibly repeated) formulas. We write $|\mathbf{A}|$ to stand for the length of $\mathbf{A}$. For any $k \in \{1, \ldots, |\mathbf{A}|\}$, by $\mathbf{A}[k]$ we mean the k-th expression in $\mathbf{A}$ and by $\mathbf{A}[1..k]$, the prefix of $\mathbf{A}$ of length k, that is, $\mathbf{A}[1] \ldots \mathbf{A}[k]$. For a binary operator $\odot \in \{\vee, \wedge, \times\}$, by $(\odot \mathbf{A})$ we mean the formula resulting from the repeated application of $\odot$ to all formulas in $\mathbf{A}$ in the same ordering. As an example, given the sequence of atoms $\mathbf{A} = (a, b, c, d, e)$, the expression $(\times \mathbf{A}[1..3])$ represents the formula $a \times b \times c$. We write $\neg \mathbf{A}$ to stand for the sequence of formulas $\neg \mathbf{A}[1] \ldots \neg \mathbf{A}[k]$ being $k = |\mathbf{A}|$. An empty conjunction is understood as $\top$ whereas an empty disjunction (both ordered or regular) is understood as $\bot$. The concatenation of two lists of formulas, $\mathbf{A}$ and $\mathbf{B}$, is simply written as $\mathbf{AB}$.

A *logic program* is a set of rules of the form:

$$(\vee \mathbf{A}) \vee (\vee \neg \mathbf{A}') \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}') \tag{7}$$

where $\mathbf{A}, \mathbf{A}', \mathbf{B}$ and $\mathbf{B}'$ are lists of atoms. We respectively call *head* and *body* to the consequent and antecedent of the implication above. A rule with an empty head $\bot$ (that is, $|\mathbf{A}| + |\mathbf{A}'| = 0$) is called a *constraint*. A rule with an empty body $\top$ (that is, $|\mathbf{B}| + |\mathbf{B}'| = 0$) is called a *fact*, and we usually write the head F instead of $F \leftarrow \top$. A rule is said to be *normal* when $|\mathbf{A}| = 1$ and $|\mathbf{A}'| = 0$. A rule is *positive* when $|\mathbf{A}'| = |\mathbf{B}'| = 0$. We extend the use of these adjectives to a program, meaning that all its rules are of the same kind.

Answer sets of a program P are defined in terms of the classical Gelfond-Lifscthiz's reduct [10], that is extended[2] as follows for the syntactic case we are considering (disjunctive heads with default negation [12]). The *reduct* of a program P with respect to a set of atoms I, written P^I, consists of a rule like $(\vee \mathbf{A}) \leftarrow (\wedge \mathbf{B})$ per each rule in P of the form (7) that satisfies $I \models (\wedge \mathbf{A}') \wedge (\wedge \neg \mathbf{B}')$. We say that a set of atoms I is an *answer set* of a program P if I is a minimal model of P^I.

Theorem 2 (From [15]). *A set of atoms T is an answer set of a program P iff $\langle T, T \rangle$ is an equilibrium model of P.* □

2.2 Logic Programs with Ordered Disjunction

A *logic program with ordered disjunction* (LPOD) is a set of rules of the form:

$$(\times \mathbf{A}) \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}') \tag{8}$$

where $\mathbf{A}, \mathbf{B}$ and $\mathbf{B}'$ are lists of atoms. We say that a set of atoms I *satisfies* an LPOD rule r like (8), written $I \models r$, when $I \models (\vee \mathbf{A}) \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}')$ in classical logic.

[2] In fact, [8] introduced a different, more general reduct that allows defining answer sets for arbitrary theories, which happen to coincide with equilibrium models. We use here the more restricted, traditional version for comparison purposes.

For each LPOD rule r like (8), we define its k-th option, written r_k, with $k \in \{1, \ldots, |\mathbf{A}|\}$, as the normal rule:

$$\mathbf{A}[k] \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}') \wedge (\wedge \neg \mathbf{A}[1..k\text{–}1])$$

A normal logic program P' is a *split program* of P if it is the result of replacing each LPOD rule $r \in P$ by one of its possible options r_k. A set of atoms I is an *answer set* of P if it is an answer set of some split program P' of P.

Example 1 (From [3]). Let P_1 be the LPOD:

$$a \times b \leftarrow \neg c \qquad\qquad b \times c \leftarrow \neg d$$

This LPOD has four split programs:

$$\begin{array}{l|l|l|l} a \leftarrow \neg c & a \leftarrow \neg c & b \leftarrow \neg c \wedge \neg a & b \leftarrow \neg c \wedge \neg a \\ b \leftarrow \neg d & c \leftarrow \neg d \wedge \neg b & b \leftarrow \neg d & c \leftarrow \neg d \wedge \neg b \end{array}$$

that yield three answer sets $\{a, b\}$, $\{c\}$ and $\{b\}$. □

As explained in [3], answer sets of LPODs can also be described in terms of a program reduct, instead of using split programs.

Definition 2 ($\times$-reduct). *The* $\times$-reduct *of an LPOD rule r like* (8) *with respect to a set of atoms I denoted as $r^I_\times$ and defined as the set of rules:*

$$\mathbf{A}[i] \leftarrow (\wedge \mathbf{B}) \tag{9}$$

for all $i = 1, \ldots, |\mathbf{A}|$ such that $I \models (\wedge \neg \mathbf{B}') \wedge (\wedge \neg \mathbf{A}[1..i\text{–}1]) \wedge \mathbf{A}[i]$. □

As expected, the $\times$-*reduct* of an LPOD P with respect to I, written $P^I_\times$ is the union of all $r^I_\times$ for all LPOD rules $r \in P$. For instance, for $I = \{b, c\}$ and P:

$$a \times b \leftarrow c \wedge \neg d \tag{10}$$

$$d \times a \leftarrow \neg b \tag{11}$$

$$d \times e \leftarrow \neg a \tag{12}$$

the reduct $P^I_\times$ would be the rule $\{b \leftarrow c\}$. Notice that $P^I_\times$ defined in this way is always a normal positive logic program and so it has a least model [20].

Theorem 3 (From [3]). *A set of atoms I is an answer set of an LPOD P iff $I \models P$ and I is the least model of $P^I_\times$.* □

It is important to note that $I \models P^I_\times$ does not imply $I \models P$, and thus, the latter is also required in the above theorem. For instance, in the last example, the interpretation $\emptyset$ is the least model of $P^I_\times$ but does not satisfy the LPOD rule (12).

Although, as said in the introduction, we will concentrate on the answer sets of split programs, we include here the definition of the three ordering relations for selecting preferred answer sets among them. We say that an LPOD rule r

like (8) is *satisfied to degree* $j \in \{1, \ldots, |\mathbf{A}|\}$ by a set of atoms I, written $I \models_j r$, when: I does not satisfy the body of r and $j = 1$; I satisfies the body of r and j is the minimum index for which $\mathbf{A}[j] \in I$. We define $deg_I(r) \stackrel{\text{def}}{=} j$ when $I \models_j r$ and define the set $I^j(P) \stackrel{\text{def}}{=} \{r \in P \mid I \models_j r\}$. Given two answer sets I, J of a given LPOD:

1. I is *cardinality-preferred* to J, written $I >_c J$, when for some truth degree k, $|I^k(P)| > |J^k(P)|$ whereas $|I^i(P)| = |J^i(P)|$ for all $i < k$.
2. I is *inclusion-preferred* to J, written $I >_i J$, when for some truth degree k, $I^k(P) \subset J^k(P)$ while $I^i(P)| = J^i(P)$ for all $i < k$.
3. I is *Pareto-preferred* to J, written $I >_p J$, if for some rule $r \in P$, $deg_I(r) < deg_J(r)$ whereas for no rule $r' \in P$, $deg_I(r') > deg_J(r')$.

3 The Ordered Disjunction Operator

Let us consider the introduction of $\times$ in **HT** as the following derived operator:

$$F \times G \stackrel{\text{def}}{=} F \vee (\neg F \wedge G) \tag{13}$$

Although in classical logic, (13) $\equiv_c F \vee G$, this equivalence does not hold in **HT**, that is (13) $\not\equiv_s F \vee G$. To see why, adding G, the two disjunctions have different equilibrium models: $\{F \vee G, G\}$ has one equilibrium model $\{G\}$, whereas $\{F \times G, G\}$ also has a second equilibrium model $\{F, G\}$. We discuss now some basic properties of $\times$ operator.

Proposition 2 (Negation). *The negation of an ordered and a regular disjunction are strongly equivalent:*

$$\neg(F \times G) \;\equiv_s\; \neg F \wedge \neg G \;\equiv_s\; \neg(F \vee G)$$

Proof. Applying De Morgan laws, $\neg(F \vee \neg F \wedge G)$ amounts to $\neg F \wedge (\neg\neg F \vee \neg G)$ which, by distributivity, is equivalent to $\neg F \wedge \neg\neg F \vee \neg F \wedge \neg G$, but the first disjunct can be removed, since $\neg F \wedge \neg\neg F$ is inconsistent in **HT** by (5). □

Proposition 3 (Truth constants). *These are some strongly equivalent simplifications:*

$$(F \times F) \equiv_s F \tag{14}$$
$$(\bot \times F) \equiv_s F \tag{15}$$
$$(F \times \bot) \equiv_s F \tag{16}$$
$$(\top \times F) \equiv_s \top \tag{17}$$
$$(F \times \top) \equiv_s (F \vee \neg F) \tag{18}$$

□

Note that the main difference with respect to ordinary disjunction when dealing with truth constants is (18). Equivalences (15) and (16) show that $\bot$ acts as a neutral element for ordered disjunction. This means that we can safely consider an empty ordered disjunction as $\bot$ (as happens with $\vee$ too).

Distributivity with respect to conjunction is satisfied in the following cases:

$$F \wedge (G \times H) \equiv_s (F \wedge G) \times (F \wedge H)$$
$$(F \times G) \wedge H \equiv_s (F \wedge H) \times (G \wedge H)$$
$$F \times (G \wedge H) \equiv_s (F \times G) \wedge (F \times H)$$

but the following pair of formulas

$$(F \wedge G) \times H \qquad\qquad (F \times H) \wedge (G \times H)$$

are not strongly equivalent. In fact, they have different answer sets. The first rule has equilibrium models $\{F, G\}$ and $\{H\}$ whereas the second has two additional ones $\{F, H\}$ and $\{G, H\}$. Distributivity between disjunctions only holds in the following case:

$$F \times (G \vee H) \equiv_s (F \times G) \vee (F \times H)$$

but the following list shows pairs of formulas that are *not* strongly equivalent:

$$F \vee (G \times H) \qquad (F \vee G) \times (F \vee H) \tag{19}$$
$$(F \times G) \vee H \qquad (F \vee H) \times (G \vee H) \tag{20}$$
$$(F \vee G) \times H \qquad (F \times H) \vee (G \times H) \tag{21}$$

Take line (19), for instance. Adding the atom H, the left rule yields two equilibrium models $\{G, H\}$ and $\{H\}$, whereas the second rule allows a third equilibrium model $\{F, H\}$. Intuitively, adding H to $F \vee (G \times H)$ will always make the second disjunct to be true (both when G holds and when it does not). So, there is no need to make F true. By a similar reason, in the case of line (20), after adding atom G, the first rule yields two equilibrium models, $\{F, G\}$ and $\{G\}$ (there is no need to make H true), while the second rule yields a third solution $\{G, H\}$. Finally, for line (21), if we add the atom H to the first expression, we get three equilibrium models, $\{F, H\}$, $\{G, H\}$, and $\{H\}$, whereas for the second expression, the addition of H only yields the equilibrium model $\{H\}$ itself.

Proposition 4 (Associativity). *The $\times$ operator is associative, that is:*

$$F \times (G \times H) \equiv_s (F \times G) \times H$$

Proof.

$$\begin{aligned}
F \times (G \times H) &\equiv_s F \vee \neg F \wedge (G \vee \neg G \wedge H) && \text{By definition of } \times. \\
&\equiv_s F \vee \neg F \wedge G \vee \neg F \wedge \neg G \wedge H && \text{Distribuitivity.} \\
&\equiv_s (F \times G) \vee \neg F \wedge \neg G \wedge H && \text{Definition of } \times. \\
&\equiv_s (F \times G) \vee \neg (F \times G) \wedge H && \text{By Proposition 2.} \\
&\equiv_s (F \times G) \times H && \text{Definition of } \times.
\end{aligned}$$

□

The third line of the proof suggests the following translation for a sequence of consecutive applications of $\times$.

Theorem 4. *Let* $\mathbf{A}$ *be a sequence of* $|\mathbf{A}| = n \geq 0$ *formulas. Then, the formula* $(\times\mathbf{A})$ *is strongly equivalent to:*

$$\bigvee_{i=1,\ldots,n} (\wedge\neg\mathbf{A}[1..i\text{–}1]) \wedge \mathbf{A}[i] \tag{22}$$

Proof. By induction on $n = |\mathbf{A}|$. When $n = 0$, both $(\times\mathbf{A})$ and (22) correspond to $\bot$. Assume it is proved for any sequence of length $n-1$, with $n > 0$. Since $\times$ is associative, we can write $(\times\mathbf{A})$ as $(\times\mathbf{A}[1..n-1]) \times \mathbf{A}[n]$. Let us call α to $(\times\mathbf{A}[1..n-1])$. Now, by definition of $\times$, $(\times\mathbf{A})$ corresponds to $\alpha \vee \neg\alpha \wedge \mathbf{A}[n]$. On the other hand, by induction, α is equivalent to:

$$\bigvee_{i=1,\ldots,n-1} (\wedge\neg\mathbf{A}[1..i\text{–}1]) \wedge \mathbf{A}[i]$$

whereas $\neg\alpha$ is equivalent to $(\wedge\neg\mathbf{A}[1..n-1])$, by Proposition 2. But then, $\alpha \vee \neg\alpha \wedge \mathbf{A}[n]$ corresponds to:

$$\underbrace{\left[\bigvee_{i=1,\ldots,n-1} (\wedge\neg\mathbf{A}[1..i\text{–}1]) \wedge \mathbf{A}[i]\right]}_{\alpha} \vee \underbrace{(\wedge\neg\mathbf{A}[1..n-1])}_{\neg\alpha} \wedge \mathbf{A}[n]$$

which is exactly the expansion in (22). □

As an example, the expression $a \times b \times c \times d$ is strongly equivalent to:

$$a \vee (\neg a \wedge b) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge \neg b \wedge \neg c \wedge d).$$

Although we have already proved that $\times$ satisfies the idempotence property (14), it must be noticed that since the operator is obviously non-commutative, a repeated subformula in a sequence of ordered disjunctions cannot always be removed. We can use Theorem 4 to prove, for instance, the following property.

Proposition 5 (Ordered Idempotence). $(F \times G \times F) \equiv_s F \times G$.

Proof. Just notice that, by Theorem 4, $(F \times G \times F)$ is equivalent to $F \vee (\neg F \wedge G) \vee (\neg F \wedge \neg G \wedge F)$, but the last disjunct can be removed due to (5). □

As F and G are arbitrary formulas, and $\times$ is associative, the equivalence above implies that, when repeated atoms occur in an ordered disjunction, all their occurrences can be removed, excepting the leftmost one. For instance:

$$a \times b \times c \times a \times d \times c \times a \times e \times b \leftrightarrow a \times b \times c \times d \times e$$

However, in the general case, we cannot remove the leftmost occurrence of a repeated formula. As a counterexample, the formulas $a \times b \times a$ and $b \times a$ are

not strongly equivalent: the simple addition of atom a to the first formula yields one equilibrium model $\{a\}$ whereas, when added to the second formula, we also obtain an additional equilibrium model $\{a, b\}$.

The following second translation is perhaps more interesting both for generating a logic program, and for comparison purposes with the $\times$-reduct. We will introduce first a useful result.

Theorem 5. *Let* $\mathbf{A}$ *be a sequence of* $|\mathbf{A}| = n \geq 0$ *formulas. Then, the formula* $(\times\mathbf{A})$ *is strongly equivalent to the conjunction of:*

$$\bigwedge_{i=1,\ldots,n} \left(\mathbf{A}[i] \vee \neg\mathbf{A}[i] \leftarrow (\wedge\neg\mathbf{A}[1..i-1]) \right) \tag{23}$$

$$\bot \leftarrow (\wedge\neg\mathbf{A}) \tag{24}$$

Proof. First, note that both formulas are classically equivalent: all conjuncts in (23) for $i = 1, \ldots, n$ are classical tautologies whereas (24) $\equiv_c (\vee\mathbf{A}) \equiv_c (\times\mathbf{A})$.

We prove now that $\langle H, T\rangle \models (\times\mathbf{A})$ iff $\langle H, T\rangle \models (23) \wedge (24)$. For the left to right direction, we begin showing that $(\times\mathbf{A})$ implies $(23) \wedge (24)$. Assume we had some $\langle H, T\rangle \models (\times\mathbf{A})$ but $\langle H, T\rangle \not\models \mathbf{A}[i] \vee \neg\mathbf{A}[i] \leftarrow (\wedge\neg\mathbf{A}[1..i-1])$ for some $i = 1, \ldots, n$. As these implications are classical tautologies, the only possibility is $\langle H, T\rangle \models (\wedge\neg\mathbf{A}[1..i-1])$ while $\langle H, T\rangle \not\models \mathbf{A}[i] \vee \neg\mathbf{A}[i]$. We will prove that, if so, $\langle H, T\rangle \not\models (22)$ which, by Theorem 4, means $\langle H, T\rangle \not\models (\times\mathbf{A})$, reaching a contradiction. Notice that we have $\langle H, T\rangle \models \neg\mathbf{A}[1], \ldots, \langle H, T\rangle \models \neg\mathbf{A}[i-1]$ while $\langle H, T\rangle \not\models \mathbf{A}[i]$ and $\langle H, T\rangle \not\models \neg\mathbf{A}[i]$. Now, the first $i-1$ disjuncts in (22) will be false because for all $j = 1, \ldots, i-1$, $\langle H, T\rangle \models \neg\mathbf{A}[j]$ implies $\langle H, T\rangle \not\models \mathbf{A}[j]$. The i-th disjunct is false because it contains $\mathbf{A}[i]$ and $\langle H, T\rangle \not\models \mathbf{A}[i]$, and the rest of disjuncts are also false since they contain the subformula $\neg\mathbf{A}[i]$ and we also had $\langle H, T\rangle \not\models \neg\mathbf{A}[i]$.

We remain to prove that $\langle H, T\rangle \models (24)$. By Proposition 2, $(\wedge\neg\mathbf{A})$ is equivalent to $\neg(\times\mathbf{A})$ and thus, (24) corresponds to $\neg\neg(\times\mathbf{A})$. But in **HT** (and in fact, in intuitionistic logic), any formula implies its double negation, and we had $\langle H, T\rangle \models (\times\mathbf{A})$.

For the right to left direction, suppose $\langle H, T\rangle \models (23) \wedge (24)$ but $\langle H, T\rangle \not\models (\times\mathbf{A})$. From the latter and Theorem 4 we conclude the following condition, let us call it (a): for all $i = 1, \ldots, n$, $\langle H, T\rangle \not\models \mathbf{A}[i]$ or there exists some j, $1 \leq j < i$ such that $\langle H, T\rangle \not\models \neg\mathbf{A}[j]$. We prove now, by induction on i in (23), that $\langle H, T\rangle \models \neg\mathbf{A}[i]$ for all $i = 1, \ldots, n$, which contradicts $\langle H, T\rangle \models (24)$. For $i = 1$ there is no j smaller and from (a) we get $\langle H, T\rangle \not\models \mathbf{A}[1]$. But, with $i = 1$, the antecedent of (23) becomes empty ($\top$) and we get $\langle H, T\rangle \models \mathbf{A}[1] \vee \neg\mathbf{A}[1]$ that, as $\langle H, T\rangle \not\models \mathbf{A}[1]$, leads to $\langle H, T\rangle \models \neg\mathbf{A}[1]$. Induction hypothesis: suppose we have proved $\langle H, T\rangle \models \neg\mathbf{A}[j]$ for all j, $1 \leq j < i$. From this together with (a) we conclude $\langle H, T\rangle \not\models \mathbf{A}[i]$. But at the same time, this induction hypothesis means that $\langle H, T\rangle$ satisfies the antecedent of (23), so we conclude $\langle H, T\rangle \models \mathbf{A}[i] \vee \neg\mathbf{A}[i]$, but again, as $\langle H, T\rangle \not\models \mathbf{A}[i]$ the only possibility is $\langle H, T\rangle \models \neg\mathbf{A}[i]$. □

The previous theorem can be combined with the following lemma to obtain a translation of any LPOD into a logic program.

Lemma 1. *If φ is strongly equivalent to a conjunction of implications $(\alpha_i \rightarrow \beta_i)$ with $i = 1, \ldots, n$ and $n \geq 0$ then the formula $F : A \rightarrow \varphi \vee B$ is strongly equivalent to G, the conjunction of $\alpha_i \wedge A \rightarrow \beta_i \vee B$.*

Proof. Take φ strongly equivalent to the conjunction of $(\alpha_i \rightarrow \beta_i)$. To prove that $F \equiv_s G$ we must show both that $F \equiv_c G$; and that $\langle H, T\rangle \models F$ iff $\langle H, T\rangle \models G$.

For proving the classical equivalence, notice that $F \equiv_c \neg A \vee \varphi \vee B$. But as strong equivalence implies classical equivalence, we can replace φ in the last formula by the conjunction of implications, obtaining $\neg A \vee B \vee \bigwedge_i (\alpha_i \rightarrow \beta_i) \equiv_c \bigwedge \neg A \vee B \vee (\alpha_i \rightarrow \beta_i) \equiv_c G$.

For proving $\langle H, T\rangle \models F$ iff $\langle H, T\rangle \models G$, assume as a first case that $\langle H, T\rangle \not\models A$ or $\langle H, T\rangle \models B$. If so, we trivially have both $\langle H, T\rangle \models F$ and $\langle H, T\rangle \models G$, and the equivalence holds. Suppose, on the contrary, that $\langle H, T\rangle \models A$ and $\langle H, T\rangle \not\models B$. It is easy to see that, then, $\langle H, T\rangle \models F$ iff $\langle H, T\rangle \models \varphi$. Similarly $\langle H, T\rangle \models G$ amounts to $\langle H, T\rangle \models \bigwedge_i (\alpha_i \rightarrow \beta_i)$ and the latter, due to our premise, is equivalent to $\langle H, T\rangle \models F$. □

In particular, we can thus translate any rule like (8) by using Lemma 1 with $H = (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}')$, $G = \bot$ and $F = (\times \mathbf{A})$ and Theorem 4 to obtain a conjunction of implications equivalent to F. In this way, for any LPOD rule r like (8) we define the set of rules r^* as the conjunction of implications in (23) and (24) after adding to all their bodies the conjunction of $(\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}')$, that is:

$$\mathbf{A}[i] \vee \neg \mathbf{A}[i] \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}') \wedge (\wedge \neg \mathbf{A}[1..i-1]) \tag{25}$$

$$\bot \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}') \wedge (\wedge \neg \mathbf{A}) \tag{26}$$

for all $i = 1, \ldots, |\mathbf{A}|$. For any LPOD P, the program P^* stands for the union of all r^* for each $r \in P$. As an example, given the LPOD rule $r : a \times b \times c \leftarrow p \wedge \neg q$, the set of rules r^* consists[3] of:

$$\begin{aligned} a \vee \neg a &\leftarrow p \wedge \neg q \\ b \vee \neg b &\leftarrow p \wedge \neg q \wedge \neg a \\ c \vee \neg c &\leftarrow p \wedge \neg q \wedge \neg a \wedge \neg b \\ \bot &\leftarrow p \wedge \neg q \wedge \neg a \wedge \neg b \wedge \neg c \end{aligned}$$

Theorem 6. *For any LPOD P, $P \equiv_s P^*$.*

Proof. It directly follows from Theorem 5 and Lemma 1. □

An important remark is that, although P^* is a disjunctive logic program with negation in the head, it belongs to a subclass of disjunctive programs with the same expressiveness and complexity than those of normal programs. To see

[3] In fact, this translation can be further refined by removing the last constraint and the last negative literal in the head. In the example, we would replace the last two rules by $c \leftarrow p \wedge \neg q \wedge \neg a \wedge \neg b$.

why, it suffices to observe that any formula of the form $F \vee \neg p \leftarrow G$ is strongly equivalent to $F \leftarrow G \wedge \neg\neg p$ and in its turn, a double negated atom in the body can be replaced[4] by the negation of a fresh auxiliary atom $F \leftarrow G \wedge \neg aux$ plus the rule $aux \leftarrow \neg p$.

Lemma 2. *Let I and J be sets of atoms and P an LPOD. Then ($J \models P^I_\times$ and $I \models P$) iff $J \models (P^*)^I$.*

Proof. Take any rule $r \in P$ of the form (8). If we apply the standard reduct with respect to I on the rules like (25) in r^*, it is easy to see that we exactly get the rules in $r^I_\times$. Thus, $P^I_\times$ coincides with the standard reduct for all the rules like (25) in P^* with respect to I. Now, let us call Q to set of remaining rules in P^* of the form (26). We will prove that $I \models P$ iff $J \models Q^I$. In fact, for each $r \in P$ like (8) we have a corresponding $r' \in Q$ like (26) and we show that $I \models r$ iff $J \models (r')^I$. We begin observing that r' (26) and r (8) are classically equivalent formulas. We have then two cases: if $I \models (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B'}) \wedge (\wedge \neg \mathbf{A})$ then r^I is $\bot$ and trivially $J \not\models \bot$, but also clearly $I \not\models r'$ and thus $I \not\models r$, and the equivalence holds. If $I \not\models (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B'}) \wedge (\wedge \neg \mathbf{A})$ the reduct $(r')^I$ is empty and thus $J \models (r')^I$ trivially, but we also have $I \models r'$ and so $I \models r$ and again the equivalence holds. □

Lemma 3. *I is an answer set of an LPOD P iff I is an answer set of P^*.* □

Proof. For the left to right direction, suppose I is an answer set of P. Then, $I \models P$ and I is a minimal model of $P^I_\times$. By Lemma 2, this implies $I \models (P^*)^I$. Suppose there exists a smaller $J \subset I$ such that $J \models (P^*)^I$. Applying Lemma 2 again, we conclude that $J \models P^I_\times$ and this contradicts that I is minimal model of $P^I_\times$.

For the right to left direction, the reasoning is analogous. If I is an answer set of P^* we obviously have $I \models (P^*)^I$ and, applying Lemma 2 we conclude $I \models P$ and $I \models P^I_\times$. Assume we have some $J \subset I$, $J \models P^I_\times$. Again, by Lemma 2 we conclude $J \models (P^*)^I$ and we get a contradiction with minimality of I as model of $(P^*)^I$. □

Theorem 7 (main theorem). *T is an answer set of an LPOD P iff $\langle T, T\rangle$ is an equilibrium model of P.*

Proof. It directly follows from Lemma 3, Theorem 6 and Theorem 2. □

As a final comment, a third possible representation of $(\times \mathbf{A})$ is the strongly equivalent formula:

$$(\vee \mathbf{A}) \wedge \bigwedge_{i=1,\ldots,n-1} \big((\vee \mathbf{A}[1..i]) \vee \neg \mathbf{A}[i] \big)$$

For instance, $a \times b \times c$ is strongly equivalent to:

$$(a \vee b \vee c) \wedge (a \vee \neg a) \wedge (a \vee b \vee \neg b) \wedge (a \vee b \vee c \vee \neg c).$$

[4] Similar techniques for removing head negations were introduced in [12, 13].

4 Disjunctive LPOD

In [14] an extension of LPODs for dealing with regular disjunction $\vee$ is considered. An *ordered disjunctive term* is defined as any arbitrary combination of atoms and $\vee, \times$ operators. A *Disjunctive LPOD* (DLPOD) is a set of rules of the form $F \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}')$ where F is an ordered disjunctive term and $\mathbf{B}, \mathbf{B}'$ are lists of atoms. An ordered disjunctive term is said to be in *Ordered Disjunctive Normal Form* (ODNF) if it has the form $(\times \mathbf{A}_1) \vee \cdots \vee (\times \mathbf{A}_n)$ where the $\mathbf{A}_i$ are lists of atoms. A DLPOD rule is in ODNF if its head is in ODNF. Similarly, the whole DLPOD is in ODNF if all its rules are in ODNF. In fact, the semantics of an arbitrary DLPOD is defined by first translating its head into ODNF using the following rewriting rules:

$$F \times (G \vee H) \longmapsto (F \times G) \vee (F \times H) \tag{27}$$
$$(F \vee G) \times H \longmapsto (F \times H) \vee (G \times H) \tag{28}$$
$$(F \times G) \times H \longmapsto F \times G \times H \tag{29}$$
$$F \times (G \times H) \longmapsto F \times G \times H \tag{30}$$

The exhaustive application of (27)-(30) allow transforming any DLPOD into ODNF, that is, a set of rules like

$$\bigvee_{i=1,\ldots,n} (\times \mathbf{A}_i) \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}') \tag{31}$$

where $n \geq 0$, and $\mathbf{B}, \mathbf{B}'$ and all the $\mathbf{A}_i$ are lists of atoms. For any DLPOD rule r like (31), an *option* for r is any rule of the form:

$$\bigvee_{i=1,\ldots,n} \mathbf{A}_i[k_i] \leftarrow (\wedge \mathbf{B}) \wedge (\wedge \neg \mathbf{B}') \wedge \bigwedge_{i=1,\ldots,n} (\wedge \neg \mathbf{A}_i[1..k_i\text{–}1]) \tag{32}$$

where each k_i is some value $k_i \in \{1, \ldots, |\mathbf{A}_i|\}$. For instance, the options for the DLPOD rule $(a \times b) \vee (c \times d \times e)$ are the rules:

$$\begin{array}{ll} a \vee c & b \vee c \leftarrow \neg a \\ a \vee d \leftarrow \neg c & b \vee d \leftarrow \neg a \wedge \neg c \\ a \vee e \leftarrow \neg c \wedge \neg d & b \vee e \leftarrow \neg a \wedge \neg c \wedge \neg d \end{array}$$

A disjunctive logic program P' is a *split program* of a DLPOD P if it is the result of replacing each DLPOD rule $r \in P$ by some of its split rules. It is important to notice that the split programs of a DLPOD are disjunctive logic programs, whereas in the case of LPODs, split programs were normal logic programs. As before, a set of atoms I is an *answer set* of a DLPOD P if it is an answer set of some split program P' of P. For comparison purposes, we concentrate here on the answer sets of DLPODs, leaving apart the selection of their preferred answer sets (see [14] for further details).

4.1 Comparison

Since our characterisation of $\times$ applies for any arbitrary propositional theory, in particular, it also provides a semantics for DLPODs, both arbitrary and in ODNF. So the immediate question is whether our **HT** based characterisation and the original DLPOD semantics coincide. Unfortunately, the answer to this question is negative, by two reasons. First, as we explained in Section 3 with a counterexample for (19), we saw that in fact, (28) is not an strongly equivalent transformation. Thus, under our **HT**-based characterisation, an arbitrary DLPOD is not always reducible to ODNF (although, of course, a semantics for the DLPOD is still defined). Still, even when we restrict ourselves to the study of ODNF programs, we can find examples where the original DLPOD semantics provides more answer sets.

Example 2. Take the DLPOD P_2 consisting of the pair of rules

$$a \vee (b \times c) \qquad c$$

The split programs of P_2 are $\{(a \vee b), c\}$ and $\{(a \vee c \leftarrow \neg b), c\}$, the first one with two answer sets $\{a, c\}$, $\{b, c\}$ and the second one with answer set $\{c\}$. In this case, there is one answer set that is *not* equilibrium model: $\{a, c\}$. □

Answer set $\{a, c\}$ looks counterintuitive in the sense that, once c is true and we are not picking choice b, it seems clear that $(b \times c)$ holds and there is no clear reason to make a true (in the first rule). Remember that $\vee$ is a regular disjunction, and it will become satisfied when any of its disjuncts become true, regardless the truth degree. The next example illustrates a counterintuitive effect of DLPOD semantics when defining subformulas with auxiliary atoms.

Example 3. Let P_3 be the DLPOD consisting of the pair of rules:

$$a \times (b \vee c) \qquad c \qquad \square$$

For both semantics, P_3 can be converted (even under strong equivalence in **HT**) into the ODNF program P_3' consisting of rule $(a \times b) \vee (a \times c)$ and fact c. Under the **HT** characterisation, both P_3 and P_3' have two answer sets, $\{a, c\}$ and $\{c\}$. For DLPOD semantics, P_3' has a third, additional answer set $\{b, c\}$. Now, notice that a different way of computing the answer sets of the original program P_3 could be defining $b \vee c$ with an auxiliary atom, so we obtain the LPOD P_3'':

$$\begin{array}{lll} a \times aux & aux \leftarrow b & b \vee c \leftarrow aux \\ c & aux \leftarrow c & \end{array}$$

It is easy to see that P_3'' has two answer sets $\{aux, a, c\}$ and $\{aux, c\}$ which, after removing the auxiliary atom, become $\{a, c\}$ and $\{c\}$, respectively, which were, in fact, the two answer sets we obtained for P_3 using the **HT** characterisation. This was expected, since **HT** satisfies the rule of substitution, that is, if $aux \leftrightarrow \alpha$ we can always replace α by aux. In these counterexamples, DLPOD semantics yielded more answer sets than the **HT** characterisation – this is actually a general property stated below[5].

[5] We leave the proof for a future extended version of this document.

Theorem 8. *If $\langle T, T\rangle$ is an equilibrium model of a DLPOD P in ODNF, then T an answer set of P.* □

5 Conclusions

We have presented a logical characterisation of Logic Programs with Ordered Disjunction (LPOD) that allows a direct study of ordered disjunction $\times$ as a derived operator in the logic of Here-and-There (**HT**), well-known for its application to (strongly equivalent) logic program transformations in Answer Set Programming. This characterisation provides an alternative implementation of LPODs that does not resort to auxiliary predicates. It has also allowed us to analyse the behavior of the $\times$ operator with respect to some typical properties like associativity, distributivity, idempotence, etc. As $\times$ is handled as a regular logical connective, our characterisation covers any arbitrary syntactic extension, and in particular, the so-called Disjunctive LPOD (DLPOD). We have shown that the semantics of DLPODs shows some differences with respect to the **HT** characterisation and established a formal comparison.

our result can also be seen as a confirmation of Theorem 6 in [6]. In that work, a reduct-based formalisation of LPODs was proposed in order to study strong equivalence relations among LPODs and regular programs. Theorem 6 in that work showed that their characterisation of strong equivalence for LPODs actually coincided with the one for regular programs. This result becomes trivial under our current approach, since ordered disjunction is just treated as an **HT** derived operator and, as proved in [4], **HT** arbitrary theories are strongly equivalent to logic programs.

We have implemented a first prototype for propositional LPODs using the current approach that uses `DLV` system as a backend[6]. Future work includes the extension of this prototype to deal with variables and with arbitrary combinations of ordered and regular disjunction in the head.

References

1. Marcello Balduccini and Veena S. Mellarkod. CR-Prolog with ordered disjunction. In *Proceedings of the 2nd Intl. on Answer Set Programming, Advances in Theory and Implementation (ASP'03)*, 2003.
2. Elisa Bertino, Alessandra Mileo, and Alessandro Provetti. PDL with preferences. In *Proc. of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 213–222, 2005.
3. Gerhard Brewka, Ilkka Niemelä, and Tommi Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, 20(2):335–357, 2004.
4. P. Cabalar and P. Ferraris. Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming*, 7(6):745–759, 2007.
5. J. P. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 20(2):308–334, 2004.

[6] See `http://www.dc.fi.udc.es/~cabalar/lpod/`

6. Wolfgang Faber, Hans Tompits, and Stefan Woltran. Notions of strong equivalence for logic programs with ordered disjunction. In Gerhard Brewka and Jérôme Lang, editors, *Proc. of the 11th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 433–443. AAAI Press, 2008.
7. P. Ferraris, J. Lee, and V. Lifschitz. A new perspective on stable models. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI'07)*, 2007.
8. Paolo Ferraris. Answer sets for propositional theories. In *Proc. of the 8th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning*, 2005. (to appear).
9. Norman Y. Foo, Thomas Meyer, and Gerhard Brewka. LPOD answer sets and Nash equilibria. In *Proc. of 9th Asian Computing Science Conference (ASIAN'04)*, pages 343–351, 2004.
10. Michael Gelfond and Vladimir Lifschitz. The stable models semantics for logic programming. In *Proc. of the 5th Intl. Conf. on Logic Programming*, pages 1070–1080, 1988.
11. Arend Heyting. Die formalen Regeln der intuitionistischen Logik. *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse*, pages 42–56, 1930.
12. Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.
13. Tommi Janhunen. On the effect of default negation on the expressiveness of disjunctive rules. In *Proc. of the 6th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, pages 93–106, 2001.
14. Philipp Kärger, Nuno Lopes, Daniel Olmedilla, and Axel Polleres. Towards logic programs with ordered and unordered disjunction. In *Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2008), 24th International Conference on Logic Programming (ICLP 2008)*, Udine, Italy, 12 2008.
15. Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.
16. Massimo Marchi, Alessandra Mileo, and Alessandro Provetti. Declarative policies for web service selection. In *Proc. of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 239–242, 2005.
17. V. Marek and M. Truszczyński. *Stable models and an alternative logic programming paradigm*, pages 169–181. 1999.
18. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.
19. David Pearce. A new logical characterisation of stable models and answer sets. In *Non monotonic extensions of logic programming. Proc. NMELP'96. (LNAI 1216)*. Springer-Verlag, 1997.
20. M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23:733–742, 1976.
21. Claudia Zepeda, Mauricio Osorio, Juan Carlos Nieves, Christine Solnon, and David Sol. Applications of preferences using answer set programming. In *In Proceedings Workshop on Answer Set Programming (ASP05)*, pages 318–332, 2005.