Preference Logic Programs using Answer Sets

Claudia Zepeda[1] and Mauricio Osorio[2]

[1] Universidad Politécnica de Puebla,
Tercer Carril del Ejido Serrano, San Mateo Cuanala,
Municipio Juan C. Bonilla, Puebla, 72640 México
czepedac@gmail.com,
[2] Universidad de las Américas, CENTIA,
Sta. Catarina Mártir, Cholula, Puebla, 72820 México
osoriomauri@gmail.com

Abstract We introduce preference rules which allow us to specify preferences as an ordering among the possible solutions of a problem. Specifically, we propose the semantics for preference logic programs. The formalism used to develop our work is Answer Set Programming(ASP). Most research on ASP and in particular about preferences in ASP supposes syntactically simple rules. So, our approach permits expressing preferences for general theories.
Keywords: Logic Programming, Answer Set Programming, Preferences.

1 Introduction

Preferences can be used to compare feasible solutions of a given problem, in order to establish if there is an order among these solutions or to establish whether such solutions are equivalents w.r.t. some requirements. Currently there are several approaches in non monotonic reasoning dealing with preferences [5]. In this paper we introduce *preference logic* (PL) programs which permit represent preferences and desires. The formalism used to develop our work is Answer Set Programming (ASP) [6]. ASP is a declarative knowledge representation and logic programming language. ASP represents a new paradigm for logic programming that allows us, using the concept of negation as failure, to handle problems with default knowledge and produce non-monotonic reasoning. Two popular software implementations to compute answer sets are DLV[1] and SMODELS[2]. The efficiency of such programs allowed to increase the list of practical applications in the areas of planning, logical agents and artificial intelligence.

Most research on ASP and in particular about preferences in ASP supposes syntactically simple rules (see for example [2,1,12]). This is justified since, most of the times, those restricted syntaxes are enough to represent a wide class of interesting and relevant problems. It could seem unnecessary to generalize the notion of answer sets to some more complicated formulas. However, a broader

[1] http://www.dbai.tuwien.ac.at/proj/dlv/
[2] http://www.tcs.hut.fi/Software/smodels/

© A. Gelbukh, S. Suárez. (Eds.)
Advances in Computer Science and Engineering.
Research in Computing Science 23, 2006, pp. 3-14

Received 07/07/06
Accepted 07/10/06
Final version 13/10/06

syntax for rules could bring some benefits. For example, the use of nested expressions could simplify the task of writing logic programs and improve their readability. Hence, PL programs is an approach about preferences and desires to general theories.

A PL program is a set of well formed formulas joined to a set of preference rules. The set of well formed formulas of a PL program allows us to obtain the different answer sets representing the solutions of a problem. For instance, $\{ice\_cream \vee (coffee \wedge cake) \leftarrow .\}$ represent two solutions, *ice-cream* and *coffee with cake*. The set of preference rules of a PL program express the preferences and desires of somebody. Preference rules use a new connective, $*$, called *preference operator* to represent an ordering among the preference options. For instance, $ice\_cream * cake \overset{pr}{\leftarrow} .$ indicates preference for ice-cream over cakes. If a preference rule has only one preference option, then it represents a desire. For instance, $coffee \overset{pr}{\leftarrow} .$ indicates the desire of drinking coffee. The complete PL program is:

$$\begin{aligned} ice\_cream \vee (coffee \wedge cake) \leftarrow . \ &\%\text{To obtain the solutions} \\ ice\_cream * cake \overset{pr}{\leftarrow} . \quad &\%\text{a preference rule} \\ coffee \overset{pr}{\leftarrow} . \quad &\%\text{a desire} \end{aligned} \tag{1}$$

Currently, there are some answer set approaches that suggest a broader syntax [8,10,4]. In particular the authors of [4] describe an approach for preferences called Answer Set Optimization (ASO) programs. ASO programs have two parts. The generating program and the preference program. The first one produce answer sets representing the solutions, and the second one expresses user preferences. The body of a rule representing preferences is defined as a conjunction of literals, and its head is defined as an ordering among the preference options. The preference options are particular formulas called boolean combinations. A boolean combination is a formula built of atoms by means of disjunction, conjunction, strong and default negation, with the restriction that strong negation is allowed to appear only in front of atoms, and default negation only in front of literals.

At this point, we could think that PL programs and ASO programs [4] could be similar approaches to represent preferences. These idea could come from the fact that both approaches have two parts (one to get the answer sets and one to express preferences), and both approaches allow us a broader syntax to express preferences. However, ASO programs and PL programs differ considerably in syntax and semantics.

As we mentioned, the first difference is related to syntax. Specifically, the parts of both approaches to get the answer sets and their parts to express preferences have different syntax. ASO programs allow us to get the answer sets from any type of logic program (for instance, normal, extended, disjunctive, etc). PL programs allows us to get the answer sets from a set of well formed formulas. Additionally, the part of ASO programs to express preferences uses boolean combinations which have a restricted syntax. The part of PL programs to express preferences uses well formed formulas. So, PL programs allow us a

broader syntax than ASO programs. The idea of permitting a broader syntax of PL programs comes from [10]. In [10] the authors propose a broader syntax for logic programs with ordered disjunction (LPOD) and its semantics. LPOD's are introduced in [2].

The second difference between ASO programs and PL programs is their semantics. For instance, we shall see in Section 3 that $\{ice\_cream\}$ and $\{coffee, cake\}$ are the preferred answer sets of the PL program (1). However, if we represent the two parts of the PL program (1) as an ASO program then, we can verify that $\{ice\_cream\}$ is its only preferred answer set. Additionally, there is only one criterion to get the preferred answer sets from an ASO program; and there are three different criteria to get the preferred answer sets from a PL program. Finally, in [4] is not defined whether ASO programs can have preferences with only one option or not. PL programs allow us to have preferences with only one option and they are called desires. We want to mention that, the semantics of PL programs is inspired by the semantics of LPOD's introduced in [2], the extended semantics for LPOD's proposed in [10], and the work about preferences in [13]. It is worth mentioning that the authors of [13] indicate that their work arose from the necessity to model a real problem. The real problem is related to represent preferences about the evacuation plans in a risk zone.

Our paper is structured as follows. In Section 2 we introduce the general syntax of the logic programs used in this paper. We also provide the definition of answer sets in terms of logic G_3. In Section 3 we present the semantics for preference logic programs. In Section 4 we present how our approach is related to extended LPOD's. Finally in Section 5 we present some conclusions.

2 Background

In this section we review some fundamental concepts and definitions that will be used along this work. We introduce first the syntax of formulas and programs based on the language of propositional logic. We also present the definition of answer sets in terms of logic G_3.

2.1 Propositional Logic

In this paper, logic programs are understood as propositional theories. We shall use the language of propositional logic in the usual way, using propositional symbols: $p, q, \ldots$, propositional connectives $\wedge, \vee, \rightarrow, \bot$ and auxiliary symbols: $(,)$. The well formed propositional formula $f \leftarrow g$ is just another way of writing $g \rightarrow f$. We assume that for any well formed propositional formula f, $\neg f$ is just an abbreviation of $f \rightarrow \bot$ and $\top$ is an abbreviation of $\bot \rightarrow \bot$. We point out that $\neg$ is the only negation used in this work. An *atom* is a propositional symbol. A *literal* is either an atom a (a positive literal) or the negation of an atom $\neg a$ (a negative literal). A *negated literal* is the negation sign $\neg$ followed by any literal, i.e. $\neg a$ or $\neg\neg a$. In particular, $f \rightarrow \bot$ is called *constraint* and it is also denoted as $\leftarrow f$. Given a set of well formed formulas F, we define $\neg F = \{\neg f \mid f \in F\}$.

Sometimes we may use *not* instead of $\neg$ and a, b instead of $a \wedge b$, following the traditional notation of logic programming. A *regular theory* or *logic program* is just a finite set of well formed formulas or rules, it can be called just *theory* or *program* where no ambiguity arises. We shall define as a *rule* any well formed formula of the form: $f \leftarrow g$. The parts on the left and on the right of " $\leftarrow$ " are called the *head* and the *body* of the rule, respectively. The signature of a logic program P, denoted as $\mathcal{L}_P$, is the set of atoms that occur in P. We want to stress the fact that in our approach, a logic program is interpreted as a propositional theory. For readers not familiar with this approach, we recommend [11,9] for further reading. We will restrict our discussion to propositional programs.

2.2 The logic G_3

Some logics can be defined in terms of truth values and evaluation functions. Gödel defined the multivalued logics G_i, with i truth values. In particular, G_2 coincides with Classical Logic C. We briefly describe in the following lines the 3-valued logic G_3 since our work uses the logical characterization of answer sets based on this logic presented in [9]. Gödel defined the logic G_3, with 3 values, with the following evaluation function I:

- $I(B \leftarrow A) = 2$ if $I(A) \leq I(B)$ and $I(B)$ otherwise.
- $I(A \vee B) = \max(I(A), I(B))$.
- $I(A \wedge B) = \min(I(A), I(B))$.
- $I(\bot) = 0$.

An interpretation is a function $I\colon \mathcal{L} \to \{0, 1, 2\}$ that assigns a truth value to each atom in the language. The interpretation of an arbitrary formula is obtained by propagating the evaluation of each connective as defined above. Recall that $\neg$ and $\top$ were introduced as abbreviations of other connectives. For a given interpretation I and a formula F we say that I is a *model* of F if $I(F) = 2$. Similarly I is a *model* of a program P if it is a model of each formula contained in P. If F is modeled by every possible interpretation we say that F is a *tautology*. For instance, we can verify that $\neg\neg a \rightarrow a$ is not a tautology in G_3, and $a \rightarrow \neg\neg a$ is a tautology in G_3. For a given set of atoms M and a program P we will write $P \vdash_{G_3} M$ to abbreviate $P \vdash_{G_3} a$ for all $a \in M$, and $P \Vdash_{G_3} M$ to denote the fact that $P \vdash_{G_3} M$ and P is consistent w.r.t. logic G_3 (i.e. there is no formula A such that $P \vdash_{G_3} A$ and $P \vdash_{G_3} \neg A$).

2.3 Answer sets

As usual in ASP, we take for granted that programs with predicate symbols are only an abbreviation of the ground program. We shall define answer sets of logic programs. The answer sets semantics was first defined in terms of the so called *Gelfond-Lifschitz reduction* [6] and it is usually studied in the context of syntax dependent transformations on programs[3]. We follow an alternative approach

[3] Currently, there are several answer set solvers, such as: DLV (http://www.dbai.tuwien.ac.at/proj/dlv/) and SMODELS(http://www.tcs.hut.fi/Software/smodels/)

started by Pearce [11] and also studied by Osorio et.al. [9]. This approach characterizes the answer sets for a propositional theory in terms of logic G_3 and it is presented in the following definition. There are several nice reasons to follow this approach. One of these reasons is that it is possible to use logic G_3 to provide a definition of ASP for arbitrary propositional theories, and at the same time to use the logic framework in an explicit way [9]. Moreover, this approach provides a natural way to extend the notion of answer sets in other logics [9]. The notation is based on [9]. We point out that $\neg$ denotes *default negation* and it is the only type of negation considered in this paper. However, it is worth mentioning that we always can handle the other negation called classical or even strong negation, denoted by $-$, by transforming the atoms with classical negation [7]. Each atom with classical negation , $-a$, that occurs in a formula of a logic program should be replaced by a new atom, a', and the rule $\neg(a \wedge a')$ should be added to the logic program. Rule $\neg(a \wedge a')$ can also be written as $(a \wedge a') \rightarrow \bot$.

Definition 1. *[9] Let P be a program and M a set of atoms. M is an* answer set *of P iff $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg\neg M \Vdash_{G_3} M$.*

For instance, the answer sets of

$$ice\_cream \vee (coffee \wedge cake) \leftarrow .$$

are $\{ice\_cream\}$ and $\{coffee, cake\}$ because,

$$P \cup \{\neg coffee, \neg cake\} \cup \{\neg\neg ice\_cream\} \Vdash_{G_3} \{ice\_cream\}, \text{ and}$$
$$P \cup \{\neg ice\_cream\} \cup \{\neg\neg coffee, \neg\neg cake\} \Vdash_{G_3} \{coffee, cake\}.$$

3 Syntax and semantics for preferences

In order to specify preferences we introduce a new connective, $*$, called *preference operator*. A preference rule specifies the preferences for something. Its head corresponds to an ordered list of well formed formulas connected using the operator $*$, where each well formed formula represents a preference option.

Definition 2. *A* preference rule *is a formula of the form: $f_1 * \cdots * f_n \overset{pr}{\leftarrow} g$ where $f_1, \ldots, f_n, g$ are well formed propositional formulas. A* preference logic (PL) program *is a finite set of preference rules and an arbitrary set of well formed formulas. If $n = 1$ the preference rule is called* desire.

If $g = \top$ the preference rule can be written as $f_1 * \cdots * f_n \overset{pr}{\leftarrow}$. The formulas $f_1 \ldots f_n$ are called the *options* of a preference rule.

Example 1. A restaurant has two options for dessert, *ice-cream* or *coffee with cake*. Peter wants ice_cream rather than cake and if possible he desires coffee. Hence, restaurant's options, and Peter's preferences and desires can be simply represented as the PL program (1), i.e.,

$$ice\_cream \vee (coffee \wedge cake) \leftarrow .$$
$$ice\_cream * cake \overset{pr}{\leftarrow} .$$
$$coffee \overset{pr}{\leftarrow} .$$

Let r_1 and r_2 be the preference rule $ice\_cream * cake \overset{pr}{\leftarrow}$. and the desire $coffee \overset{pr}{\leftarrow}$. respectively.

The answer sets of a PL program are the answer sets of the logic program obtained by remove the preference rules from the original PL program.

Definition 3. *Let $Pref$ be the set of preference rules of a PL program P. Let M be a set of atoms. M is an answer set of P iff M is an answer set*[4] *of $P \setminus Pref$.*

So, the answer sets of the PL program of Example 1 are $\{ice\_cream\}$ and $\{coffee, cake\}$.

Part of the semantics of PL programs was inspired by the semantics of LPOD's [2]. Due to lack of space we do not present the semantics of LPOD's, but readers not familiar with this approach can review [2]. The semantics of PL programs is based on a function called *satisfaction degree*. The satisfaction degree with respect of an answer set of a preference rule indicates how well is this preference rule satisfied by the answer set. A satisfaction degree equal to 1 is better than all others. A satisfaction degree lower than other is better than the second one. Our definition of satisfaction degree is in terms of logic G_3, however since logic programs (or theories) used in this work are complete (i.e. for any formula A of a program P, either $P \vdash_{G_3} A$ or $P \vdash_{G_3} \neg A$), we could use classic logic too [5].

Definition 4. *Let M be an answer set of a PL program P. Let $r := f_1 * \cdots * f_n \overset{pr}{\leftarrow} g$ be a preference rule of P. Let m be the $max\{n \mid f_1 * \cdots * f_n \overset{pr}{\leftarrow} g$ is a preference rule of $P\}$. We define the* satisfaction degree of r *in M, denoted by $deg_M(r)$, as a correspondence rule that defines the following function:*

1. 1 *if* $M \cup \neg(\mathcal{L}_P \setminus M) \nvdash_{G_3} g$.
2. $min\{i \mid M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{G_3} f_i\}$ *if* $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{G_3} g$.
3. $m + 1$ *if* $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{G_3} g$ *and there is not* $1 \leq i \leq n$ *such that* $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{G_3} f_i$.

The part (1) of Definition 4 indicates that rule r has a satisfaction degree equal to 1 in the answer set because, the rule r does not apply and this fact makes rule r irrelevant to prefer the answer set. The part (2) of Definition 4 indicates that rule r is satisfied in the answer set to some degree. Finally, the part (3) of Definition 4 indicates that the rule r has the largest value of satisfaction degree in the answer set because non of the options of the preference rule r holds. According to our intuition, this part (3) of Definition 4 will be useful in case

[4] Note that since we are not considering strong negation, there is no possibility of having inconsistent answer sets.

[5] For complete theories, logic G_3 is equivalent to classic logic [9].

that non of the options of each preference rule in P holds in all the answer sets of P. So, in this case all the answer sets of the PL program should be preferred. For instance, let us suppose that I have a preference for fruit over cookies and orange juice over milk for breakfast. Additionally, I have only two options for breakfast, salad or eggs. In this case both options are incomparable with respect to my preferences and both of them should be preferred.

Example 2. Let P be the PL program of Example 1. By Definition 3 we know that program P has two answer sets: $M_1 = \{ice\_cream\}$ and $M_2 = \{coffee, cake\}$. According to Definition 4 we can verify that $m = 2$ and that,

$$\begin{array}{ll} deg_{M_1}(r_1) = 1, & deg_{M_2}(r_1) = 2, \\ deg_{M_1}(r_2) = 3, & deg_{M_2}(r_2) = 1. \end{array}$$

It is interesting to mention that $deg_{M_1}(r_2)$ is equal to 3 because, non of the options of r_2 holds in M_1 (see part (3) of Definition 4). So, $deg_{M_1}(r_2) = m + 1$.

The following theorems and definitions are about the preferred answer sets of a PL program. All of them are similar to the definitions given in [2]. However we do not have to forget that they are defined for general theories (see Definition 2) and are based on our own definition of satisfaction degree.

Theorem 1. *Let $Pref$ be the set of preference rules of a PL program P. If M is an answer set of P then M satisfies all the rules in $Pref$ to some degree.*

The satisfaction degree of each preference rule of a PL program allows us to define the set of preference rules with the same satisfaction degree. We use these sets to obtain the preferred answer sets of the PL program according to some criterion.

Definition 5. *Let P be a PL program and let $Pref$ be the set of preference rules of P. Let M an answer set of P. We define $S^i_M(P) = \{r \in Pref \mid deg_M(r) = i\}$.*

Example 3. Let P be the PL program of Example 1. Let us consider the satisfaction degree of rules r_1 and r_2 in Example 2. Then we can verify that,

$$\begin{array}{lll} S^1_{M_1}(P) = \{r_1\}, & S^2_{M_1}(P) = \{\}, & S^3_{M_1}(P) = \{r_2\}, \\ S^1_{M_2}(P) = \{r_2\}, & S^2_{M_2}(P) = \{r_1\}, & S^3_{M_2}(P) = \{\}. \end{array}$$

The following definitions indicate how to apply different criteria to the sets of preference rules $S^i_M(P)$ of a PL program, in order to know if one answer set is preferred to another answer set and to obtain the most preferred answer sets. The criteria used are *set inclusion*, *set cardinality* or *pareto criterion*. We start describing how to apply the set inclusion criterion.

Definition 6. *Let M and N be answer sets of a PL program P. M is* inclusion preferred *to N, denoted as $M >_i N$, iff there is an i such that $S^i_N(P) \subset S^i_M(P)$ and for all $j < i$, $S^j_M(P) = S^j_N(P)$.*

Definition 7. *A set of atoms M is an* inclusion-preferred *answer set of a PL program P, if M is an answer set of P and there is not answer set M' of P, $M \neq M'$, such that $M' >_i M$.*

Example 4. Let P be the PL program of Example 1. If we consider the results of Example 3 then, we can verify that M_1 is not inclusion-preferred to M_2 or vice versa since $S^1_{M_1}(P)$ is not a subset of $S^1_{M_2}(P)$ or vice versa. We also can see that there is not M answer set of P, $M \neq M_1$ and $M \neq M_2$, such that $M >_i M_1$ or $M >_i M_2$. Hence M_1 and M_2 are both the inclusion-preferred answer sets of P.

From Example 4 we can see that if M_1 and M_2 are inclusion-preferred answer sets of P then, it means that Peter could have *ice-cream* or *coffee with cakes* for breakfast. This result agrees with our intuition because according to Peter's preferences and desires: M_1 includes *ice-cream* that is one of the options with the highest preference, and M_2 includes *coffee* that is a desire with the highest preference too.

The following two definitions indicate how to use set cardinality criterion to prefer an answer set to another answer set, and to obtain the most preferred answer sets. Before presenting these definitions, we have to mention that this criterion was particularly useful to specify preferences for evacuation plans using ASP approaches in [13]. One of the criteria used to prefer the evacuation paths in [13] was the number of segments of roads in each evacuation path. In this case, the paths with the minimum number of segments of road were the preferred paths. So, the idea of using the cardinality set criterion resulted very natural and easy in this case.

Definition 8. *Let M and N be answer sets of a PL program P. M is* cardinality preferred *to N, denoted as $M >_c N$, iff there is an i such that $|S^i_M(P)| > |S^i_N(P)|$ and for all $j < i$, $|S^j_M(P)| = |S^j_N(P)|$.*

Definition 9. *A set of atoms M is a* cardinality-preferred *answer set of a PL program P, if M is an answer set of P and there is not answer set M' of P, $M \neq M'$, such that $M' >_c M$.*

Example 5. Let P be the PL program of Example 1. If we consider the results of Example 3 then, we can verify that M_2 is cardinality-preferred to M_1 since $|S^2_{M_2}(P)| > |S^2_{M_1}(P)|$ and $|S^1_{M_2}(P)| = |S^1_{M_1}(P)|$. We also can see that there is not answer set M' of P, $M' \neq M_2$, such that $M' >_c M_2$. Hence M_2 is the cardinality-preferred answer set of P.

From the Example 5 we can see that if the criterion to prefer one of the menu options for breakfast is the number of things that the option includes then, we agree that $M_2 := \{coffee, cakes\}$ is a better option than $M_1 := \{ice\_cream\}$ because M_2 has two things to eat and M_1 has only one.

Finally, we describe the pareto criterion. As it is described in [2], in some cases the result of adding not achievable options to preference rules does not agrees with what we could expect. For instance, let us extend the problem specification of Example 1 as follows: the restaurant also can offer gelatine in the mornings, Peter prefers gelatine to ice_cream, and Peter can't have gelatine because he always arrives at the restaurant at night. So, restaurant's options and Peter's preferences can be represented as the following PL program,

$$gelatine \vee ice\_cream \vee (coffee \wedge cake) \leftarrow .$$
$$\leftarrow gelatine.$$
$$gelatine * ice\_cream * cake \overset{pr}{\leftarrow} .$$
$$coffee \overset{pr}{\leftarrow} .$$

In spite of our intuition indicates that this program should have the same two inclusion-preferred answer sets of Example 4, we can verify that $\{coffee, cake\}$ is the only inclusion preferred-answer set of this PL program. In order to avoid effects of this kind we can use the pareto criterion:

Definition 10. *Let M and N be answer sets of a PL program P. Let $Pref$ be the set of preference rules of P. M is* pareto preferred *to N, denoted as $M >_p N$, iff there is an $r \in Pref$, such that $deg_M(r) < deg_N(r)$, and for no $r' \in Pref$ $deg_N(r') < deg_M(r')$.*

Definition 11. *A set of atoms M is a* pareto-preferred *answer set of a PL program P, if M is an answer set of P and there is not answer set M' of P, $M \neq M'$, such that $M' >_p M$.*

Example 6. Let P be the PL program of Example 1. Let us consider the results of Example 2. We can verify that $deg_{M_1}(r_1) < deg_{M_2}(r_1)$ and $deg_{M_2}(r_2) < deg_{M_1}(r_1)$. So, M_1 is not pareto-preferred to M_2. In a similar way we can verify that M_2 is not pareto-preferred to M_1. We also can see that there is not M answer set of P, $M \neq M_1$ and $M \neq M_2$, such that $M >_p M_1$ or $M >_p M_2$. Hence M_1 and M_2 are both the pareto-preferred answer sets of P.

4 Related work

Currently there are several approaches in non monotonic reasoning dealing with preferences [5]. Balduccini et al. relate ASP with preferences introducing CR-programs with preferences in [1]. Work that relates Answer Set Planning with preferences using language $\mathcal{PP}$ can be found in [12]. Work that relates ASP with preferences as an LPOD can be found in [2]. LPOD's are a useful extension of ASP, providing more natural modeling and easier solutions to many problems. Currently it is possible to compute the preferred answer sets under the ordered disjunction semantics using *Psmodels* [3]. *Psmodels* is a modification of SMODELS to compute the preferred answer sets of LPOD's. In [10] a broader syntax for LPOD's and its semantics is proposed. This approach was called extended logic programs with ordered disjunction (ELPOD).

The approach presented in this paper follows the approaches given in [2,10]. In spite of ELPOD's [10] and PL programs have a similar broader syntax and use the three criteria (set inclusion, set cardinality and pareto criterion) to get the preferred answer sets, both approaches are different. Specifically, PL programs differ from ELPOD's in two main features.

The first difference between PL programs and ELPOD's is their semantics. ELPOD's represent a particular prioritized form of disjunction over the preference options and PL programs represent preference over the preference options. For

instance, if we represent the two parts of the PL program of Example 1 as an ELPOD then, we can verify that $\{ice\_cream, coffee\}$ and $\{ice\_cream, coffee, cake\}$ are the preferred answer sets using the three criteria: *set inclusion*, *set cardinality* and *pareto criterion*. Clearly, this result does not agree with our intuition, results and discussions presented in Example 4, Example 5, and Example 6.

The second difference between PL programs and ELPOD's is the form to represent a problem with preferences. A PL program represents this kind of problems as a set of well formed formulas joined to a set of preference rules. So a PL program has two parts, one to generate the answer sets and one to express the preferences. In ELPOD's this does not occur. An ELPOD represents a problem with preferences as a set of extended ordered disjunction rules.

On the other hand, in [14] the authors define the concept of maximal answer sets of a program w.r.t. a set of atoms. For instance, if the answer sets of a program P are $\{b, c, e\}$, $\{b, c, d\}$, and $\{e, a, c\}$ then, $\{b, c, d\}$ is the maximal answer set with respect the set of atoms $A := \{b, d, f\}$ because it has the maximum number of elements in the set A. Moreover, in [14] is described how this concept could be useful in a real application related to argumentation in the domain of organ transplantation. The idea in [14] is to get the maximal answer sets from a particular program such that, these maximal answer sets correspond to the preferred extensions of an argument framework.

It is easy to verify that PL programs can be used to obtain the maximal answer sets of a logic program P w.r.t. a set of atoms A. The PL program used is obtained by adding to P a set of *desires* (see Definition 2). The set of desires is defined as follows: for each atom a in A, the desire $a \overset{pr}{\leftarrow} .$ is added. So, in the previous example the PL program used to obtain the maximal answer sets of P w.r.t. A is: $P \cup \{b \overset{pr}{\leftarrow} .,\ \ d \overset{pr}{\leftarrow} .\ \ f \overset{pr}{\leftarrow} .\}$. The intuition behind each desire is to indicate that we want that each atom a in A is in the answer sets of the original program P.

Finally, we want to mention that if we use a PL program P' to compute the maximal answer sets of a program P w.r.t. a set of atoms A then, it is possible to translate P' to a particular ELPOD P'' and P'' to a LPOD P'''. So, it is possible to use *Psmodels* [3] with the inclusion set criterion to compute the maximal answer sets of P from P'''.

The translation of the PL program P' to an ELPOD P'' is as follows: each desire $a \overset{pr}{\leftarrow}$ in P' has to be replaced by the extended ordered rule $\neg\neg a \times a^{\bullet}$ where $a^{\bullet}$ is an atom that neither occurs in P' nor occurs in A. Since $\neg\neg a$ is equivalent to the restriction $\leftarrow \neg a$, the intuition behind $\neg\neg a$ is to indicate that a should be in the model of a program. We can find the details about why to use double default negation in [14].

The translation of the ELPOD P'' to a standard LPOD P''' is due to the fact that neither running *Psmodels* [3] nor following the definition given by Brewka [2] for LPOD's we can obtain the inclusion preferred answer sets for ELPOD's. The reason is that the definition given by Brewka for ordered disjunction has syntactical restrictions. So, in [14] it is also indicated how to translate the par-

ticular ELPOD P'' to a standard LPOD P'''. It is worth to mention that this translation is very simple.

5 Conclusions

In this work we provide two semantics for PL programs. We propose the only ASP approach about preferences and desires that allows us to express preference rules for general theories. We discuss why a broader syntax for rules could bring some benefits. We also present how our approach is related to ELPOD's and how it differs from ELPOD's.

In future work we plan to research about the properties of the semantics of preference logic programs.

References

1. M. Balduccini and V. S. Mellarkod. A-prolog with cr-rules and ordered disjunction. In *International Conference on Intelligent Sensing and Information Processing*, pages 1–6, 2004.
2. G. Brewka. Logic Programming with Ordered Disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence, AAAI-2002.* Morgan Kaufmann, 2002.
3. G. Brewka, I. Niemelä, and T. Syrjänen. Implementing Ordered Disjunction Using Answer Set Solvers for Normal Programs. In *Proceedings of the 8th European Workshop Logic in Artificial Inteligence JELIA 2002.* Springer, 2002.
4. G. Brewka, I. Niemela, and M. Truszczynski. AnswerSet Optimization. In *IJCAI-03*, pages 867–872, 2003.
5. J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 2004.
6. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.
7. M. Gelfond and V. Lifschitz. Logic Program with Classical Negation. In D. H. D. Warren and P. Szeredi, editors, *Proceedings of the 7th Int. Conf. on Logic Programming*, pages 579–597, Jerusalem, Israel, June 1990. MIT.
8. N. Leone and S. Perri. Parametric Connectives in Disjunctive Logic Programming. In *ASP03 Answer Set Programming: Advances in Theory and Implementation*, Messina, Sicily, Sept. 2003.
9. M. Osorio, J. A. Navarro, and J. Arrazola. Applications of Intuitionistic Logic in Answer Set Programming. *Theory and Practice of Logic Programming (TPLP)*, 4:325–354, May 2004.
10. M. Osorio, M. Ortiz, and C. Zepeda. Using CR-rules for evacuation planning. In G. D. I. Luna, O. F. Chaves, and M. O. Galindo, editors, *IX Ibero-american Workshops on Artificial Inteligence*, pages 56–63, 2004.
11. D. Pearce. Stable Inference as Intuitionistic Validity. *Logic Programming*, 38:79–91, 1999.
12. T. C. Son and E. Pontelli. Planning with preferences using logic programming. In *LPNMR*, pages 247–260, 2004.

13. C. Zepeda. *Evacuation Planning using Answer Sets.* PhD thesis, Universidad de las Americas, Puebla and Institut National des Sciences Appliquées de Lyon, 2005.
14. C. Zepeda, M. Osorio, J. C. Nieves, C. Solnon, and D. Sol. Applications of preferences using answer set programming. In *Submmited to Answer Set Programming: Advances in Theory and Implementation (ASP 2005).*