# A general framework for representing preferences

Claudia Zepeda, José Luis Carballido, and Sergio Arzola

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
{czepedac,jlcarballido7}@gmail.com

**Abstract.** In this paper we propose a framework for representing preference problems. The proposal is based on a logic preference program composed by two parts: the *generator part* and the *preference part.* The fist part generates the set of alternative solutions and the second part express the preferences used to obtain the preferred solutions. Moreover, our approach can use one of two logic programming semantics to obtain the solutions from the generator part.
**Key words:** Preferences, logic programming semantics.

## 1 Introduction

Commonly when we model a problem we obtain a set of alternative solutions, then preferences are useful to make an election among these different alternatives. In this respect, we propose a general approach to model preference problems where we define two parts: the *generator part* that generates the set of alternative solutions and the *preference part* that express the preferences used to obtain the preferred solutions among the set of alternative solutions.

Our approach has the advantage of defining the *generator part* as a logic program that corresponds to the formulation of a problem solving task which generates the set of alternative solutions of a given problem. Moreover, the set of alternative solutions can be obtained using the different semantics such as the *stable semantics* [9] or the *p-stable semantics* [14].

It is natural to consider the stable semantics since many preference approaches have been based on it, see for example [5, 15, 3]. On the other hand, the p-stable semantics has the advantage of providing models that coincide with classical models in many cases [14]. Besides, it has been shown that the p-stable semantics of normal programs can express any problem that can be expressed in terms of the stable semantics of disjunctive programs in [13].

The *preference part* allows us to define an order on the set of alternative solutions of a given problem. Once the alternatives solutions are ordered we can define the preferred solution as the minimal or maximal solution.

The paper is structured as follows. In Section 2, we introduce some fundamental definitions about logic programming and logic programming semantics. In Section 3 we define our general approach to model preference problems. In

Section 4 we describe an example of our proposed approach. Finally in Section 5 we present some conclusions.

## 2 Background

In this section, we define the syntax of the logic programs that we will use in this paper. In terms of logic programming semantics, we present the definition of the stable model semantics and the p-stable model semantics.

### 2.1 Logic programs

We use the language of propositional logic in the usual way. We consider *propositional symbols*: $p, q, \ldots$; *propositional connectives*: $\wedge, \vee, \rightarrow, \neg, -$; and *auxiliary symbols*: '(','),'.'. Well formed propositional formulas are defined as usual. We consider two types of negation: strong or classical negation (written as $-$) and negation-as-failure (written as $\neg$). Intuitively, $\neg a$ is true whenever there is no reason to believe $a$, whereas $-a$ requires a proof of the negated atom. An *atom* is a propositional symbol. A *literal* is either an atom $a$ or the strong negation of an atom $-a$.

A *normal* clause is a clause of the form $a \leftarrow b_1 \wedge \ldots \wedge b_n \wedge \neg b_{n+1} \wedge \ldots \wedge \neg b_{n+m}$ where $a$ and each of the $b_i$ are atoms for $1 \leq i \leq n + m$. In a slight abuse of notation we will denote such a clause by the formula $a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$ where the set $\{b_1, \ldots, b_n\}$ will be denoted by $\mathcal{B}^+$, and the set $\{b_{n+1}, \ldots, b_{n+m}\}$ will be denoted by $\mathcal{B}^-$. Given a normal clause $a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$, denoted by $r$, we say that $a = H(r)$ is the *head* and $\mathcal{B}^+(r) \cup \neg\mathcal{B}^-(r)$ is the *body* of the clause.

A clause with an empty body is called a *fact*; and a clause with an empty head is called a *constraint*. Facts and constraints are also denoted as $a \leftarrow$ and $\leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$ respectively. We define a *normal logic program* $P$, as a finite set of normal clauses. The signature of a normal logic program $P$, denoted as $\mathcal{L}_P$, is the set of atoms that occur in $P$. Given a set of atoms $M$ and a signature $\mathcal{L}$, we define $\neg\widetilde{M} = \{\neg a \mid a \in \mathcal{L} \setminus M\}$. Since we shall restrict our discussion to propositional programs, we take for granted that programs with predicate symbols are only an abbreviation of the ground program. From now on, by *logic program* or *program* we will mean a normal logic program when ambiguity does not arise.

In order to accept literals in our discussion we will manage the strong negation $-$ as follows: each atom $-a$ is replaced by a new atom symbol $a'$ which does not appear in the language of the program and we add the constraint $\leftarrow a \wedge a'$ to the program.

### 2.2 Logic programming semantics

Here, we present the definitions of three logic programming semantics. Note that we only consider 2-valued logic programming semantics.

**Definition 1.** *A logic programming semantics $S$ is a mapping that assigns to a program $P$ a subset of $2^{\mathcal{L}_P}$.*

We sometimes refer to *logic programming semantics* as *semantics*, when no ambiguity arises. The semantics that we consider in this paper are: *the stable model semantics* [9] (denoted by *stable*), and *the p-stable model semantics* [14] (denoted by *p-stable*). From now on, we assume that the reader is familiar with the notion of an *interpretation* and *validity* [16].

**Stable semantics** The stable semantics was defined in terms of the so called *Gelfond-Lifschitz reduction* [9] and it is usually studied in the context of syntax dependent transformations on programs. The following definition of a stable model for normal programs was presented in [9].

**Definition 2.** *Let* P *be any program. For any set $M \subseteq \mathcal{L}_P$, let $P^M$ be the definite program obtained from* P *by deleting each rule that has a literal $\neg l$ in its body with $l \in M$, and then all literals $\neg l$ in the bodies of the remaining clauses. Clearly $P^M$ does not contain $\neg$, then* M *is a stable model of* P *if and only if* M *is a minimal model of $P^M$.*

*Example 1.* Let $M = \{b\}$ and $P$ be the following program: $\{b \leftarrow \neg a, \quad c \leftarrow \neg b, \quad b \leftarrow, \quad c \leftarrow a\}$. Notice that $P^M$ has three models: $\{b\}$, $\{b, c\}$ and $\{a, b, c\}$. Since the minimal model among these models is $\{b\}$, we can say that $M$ is a stable model of $P$.

**p-stable semantics** Before defining the p-stable semantics (introduced in [14]), we define some basic concepts. Logical inference in classic logic is denoted by $\vdash$. Given a set of proposition symbols $S$ and a theory (a set of well-formed formulas) $\Gamma$, $\Gamma \vdash S$ if and only if $\forall s \in S$, $\Gamma \vdash s$. When we treat a program as a theory, each negative literal $\neg a$ is regarded as the standard negation operator in classical logic. Given a normal program P, if $M \subseteq \mathcal{L}_P$, we write $P \Vdash M$ when: $P \vdash M$ and $M$ is a classical 2-valued model of $P$.

The p-stable semantics is defined in terms of a single reduction which is defined as follows:

**Definition 3.** *[14] Let $P$ be a program and $M$ be a set of literals. We define*

$$RED(P, M) = \{a \leftarrow \mathcal{B}^+ \cup \neg(\mathcal{B}^- \cap M) \mid a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^- \in P\}$$

*Example 2.* Let us consider the program $P_1 = \{a \leftarrow \neg b \wedge \neg c, \quad a \leftarrow b, \quad b \leftarrow a\}$ and the set of atoms $M_1 = \{a, b\}$. We can see that $RED(P, M)$ is: $\{a \leftarrow \neg b, \quad a \leftarrow b, \quad b \leftarrow a\}$.

Next we present the definition of the p-stable semantics for normal programs.

**Definition 4.** *[14] Let $P$ be a program and $M$ be a set of atoms. We say that $M$ is a* p-stable model *of $P$ if $RED(P, M) \Vdash M$. We use p-stable to denote the semantics operator of p-stable models.*

*Example 3.* Let us consider again $P_1$ and $M_1$ of Example 2. Let us verify whether $M_1$ is a p-stable model of $P_1$. First, we can see that $M_1$ is a model of $P_1$, i.e., for each clause $C$ of $P_1$, $M_1$ evaluates $C$ to true. We also can verify that $RED(P_1, M_1) \vdash M_1$ . Then we can conclude that $RED(P_1, M_1) \Vdash M_1$. Hence, $M_1$ is a *p-stable model* of $P_1$.

The following examples illustrate how to obtain the p-stable models. The first example shows a program with a single p-stable model, which is also a classical model. The second example shows a program which has no stable models and whose p-stable and classical models are the same.

*Example 4.* Let $P = \{q \leftarrow \neg q\}$. Let us take $M = \{q\}$ then $RED(P, M) = \{q \leftarrow \neg q\}$. It is clear that $M$ models $P$ in classical logic and $RED(P, M) \models M$ since $(\neg q \rightarrow q) \rightarrow q$ is a theorem in classical logic with the negation $\neg$, now interpreted as classical negation. Therefore $M$ is a p-stable model of $P$.

*Example 5.* Let $P = \{a \leftarrow \neg b, \;\; a \leftarrow b, \;\; b \leftarrow a\}$. We can verify that $M = \{a, b\}$ models the clauses of $P$ in classical logic. We find that $RED(P, M) = P$. Now, from the first and third clause, it follows that $(\neg b \rightarrow b)$ where the negation $\neg$ is now interpreted as classical negation. Since $(\neg b \rightarrow b) \rightarrow b$ is a theorem in classical logic, it follows that $RED(P, M) \models M$. Therefore, $M$ is a p-stable model of $P$.

It is worth mentioning that there exists also a characterization of the p-stable semantics in terms of the paraconsistent logic $G'_3$, interested readers can see [13, 14].

## 3 General approach for defining preferences

Now we define our approach to model preference problems. The proposed approach defines a preference program that can be seen as the union of a logic program called the *generator-program* that corresponds to the *generator part* and a set of preference rules that corresponds to the *preference part.* From the generator part we obtain the set of alternative solutions based on a logic programming semantics. From the preference part we obtain the preferred solution. Now, we give a description of each part.

A *generator-program* is a normal logic program that corresponds to the formulation of a problem solving task which generates the set of alternative solutions of a given problem. This logic program could be obtained following some methodology for declarative problem solving, for example the methodology called generate and test (see [2]), it also could be a program obtained according to an existing approach to model update problems (such as [1, 7]), argumentation problems (such as [6, 12, 4, 11]), planning problems (such as [10, 8, 2] etc.).

*Example 6.* Let $P$ be the following normal logic program which is an example of generator program.

$$\begin{array}{l} a \leftarrow . \\ c \leftarrow \neg b. \\ b \leftarrow \neg c. \end{array}$$

Given a generator logic program $P$ and a logic programming semantics $X$, we are going to denote as $X$-*model* a stable model or a p-stable model, depending on the semantics $X$.

*Example 7.* Let us consider the generator logic program $P$ of Example 6 and let $X$ be the stable semantics. Hence the stable-models of $P$ are $M_1 = \{a, b\}$ and $M_2 = \{a, c\}$.

On the other hand, in order to specify preferences we introduce a new connective, $*$, called *preference operator.* A preference rule specifies the preferences for something.

**Definition 5.** *A* preference rule *is a formula of the form:* $p_1 * \cdots * p_n$ *where* $p_1, \ldots, p_n$ *are atoms called* options*.*

In a preference rule, the most preferred option is the first left and the less preferred option is the last to the right.

Now, we present the definition of a preference program.

**Definition 6.** *We say that $P$ is a* preference program *if it corresponds to the union of a* generator-program *denoted as* $Gen(P)$*, and a set of preference rules denoted as* $Pref(P)$*.*

*Example 8.* Let $P$ be the following preference program.

$$\begin{array}{l} a * b. \\ a \leftarrow \neg c. \\ b \leftarrow \neg d. \end{array}$$

$Gen(P)$ corresponds to the second and third rules of $P$, while $Pref(P)$ corresponds to the first rule of $P$.

Now, we define the X-models of a preference program.

**Definition 7.** *Let $P$ be a preference program. We say that a set of atoms $M$ is an X-model of $P$ if $M$ is an X-model of* $Gen(P)$*.*

Therefore the X-models of the preference program $P$ of Example 8 are $M_1 = \{a, b\}$ and $M_2 = \{a, c\}$ according to the Example 7.

The preference rules of a preference program allow us to define an order on the set of alternative solutions of a given problem and then obtain the preferred solution. Formally, given a preference program $P$, a partial order $<_P$ based on the set of preference rules $Pref(P)$ gives a partial order on the set of $X$-*models* of $P$. Consequently, the minimal (or maximal) X-models will be the preferred X-models, according to the given partial order. In the next section we describe an example of our approach proposed, where we define a particular partial order and the stable semantics is used.

## 4 An instance of our preference approach

Here, we describe an example of our proposed approach. We define formally, the two parts of a preference program, namely the Generator-Program and the preference rules. Besides, we define a particular partial order to obtain the preferred models.

The *generator-Program* is a normal logic program that corresponds to the formulation of a problem solving task which generates the set of alternative solutions of a given problem.

A *preference rule* is a formula of the form $p_1 * \cdots * p_n$ where $p_1, \ldots, p_n$ are atoms.

Here, we say that $P$ is a *preference program* if it corresponds to the union of a *generator-program* denoted as $Gen(P)$, and one preference rule denoted as $pref(P)$.

Now we define the partial order to obtain the most preferred X-model of a preference program based on the preference rule.

**Definition 8.** *Let $P$ be a preference program, and let $M$ and $N$ be two X-models of $P$. Let $p_1 * \cdots * p_n$ be the preference rule of $P$. The X-model $M$* is preferred to the X-model *$N$, denoted as $N < M$, if*

1. *there exists $i = min(1 \leq k \leq n)$ such that $p_i \in M$ and $p_i \notin N$, and*
2. *for all $j < i$, ($p_j \in M$ and $p_j \in N$) or ($p_j \notin M$ and $p_j \notin N$).*

Now, we provide the definition of the most preferred X-model of a preference program.

**Definition 9.** *Given an X-model $M$ of a normal program $P$, we say that $M$ is* the most preferred X-model *of $P$ if there is no other X-model $N$ of $P$ such that $M < N$.*

*Example 9.* Let $X$ be the stable semantics. Let $P$ be the following preferred program:

$$\begin{array}{l} b * c. \\ a \leftarrow . \\ c \leftarrow \neg b. \\ b \leftarrow \neg c. \end{array}$$

We can verify that $P$ has two stable-models, $\{a, b\}$ and $\{a, c\}$. We also can verify that the stable-model $\{a, b\}$ *is preferred to the stable-model* $\{a, c\}$, i.e., $\{a, c\} < \{a, b\}$ and that $\{a, b\}$ is also *the most preferred stable-model* of $P$. When $X$ is the p-stable semantics we can also verify that the p-stable models and the most preferred p-stable-model coincide with the stable models and the most preferred stable-model.

## 5 Conclusion

When we model a problem we can obtain a set of alternative solutions, then preferences can be useful to find feasible solutions. We proposed an approach for representing preference problems. The proposal is based on a logic preference program composed by two parts: the *generator part* and the *preference part*. Our approach can use the p-stable semantics or the stable semantics to obtain the set of alternative solutions from the generator part.

## Acknowledgement

This research has been supported by the Fondo Sectorial SEP-CONACyT, Ciencia Básica Project (Register 101581).

## References

1. F. Banti, J. J. Alferes, and A. Brogi. A principled semantics for logic program updates. In *Workshop on Nonmonotonic Reasoning, Action and Change. Eighteenth International Joint Conference on Artificial Intelligence*, 2003.
2. Chitta Baral. *Knowledge Representation, reasoning and declarative problem solving with Answer Sets.* Cambridge University Press, Cambridge, 2003.
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)*, pages 444–455, Cosenza, Italy, 2002.
4. José Luis Carballido, Juan Carlos Nieves, and Mauricio Osorio. Inferring Preferred Extensions by Pstable Semantics. *Revista Iberomericana de Inteligencia Artificial*, 13(41):38–53, 2009.
5. J. 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. Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.
7. T. Eiter, M. Fink, G. Sabattini, and H. Thompits. Considerations on Updates of Logic Programs. In Manuel Ojeda-Aciego and Luís Moniz Pereira Inman P. de Guzmán, Gerhard Brewka, editors, *Logics in Artificial Intelligence, European Workshop, JELIA 2000.*, Malaga, Spain, September 2000. Springer Verlag.
8. Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. Planning under incomplete knowledge. In *Proceedings of the First International Conference on Computational Logic*, pages 807–821, London, UK, July 2000. Springer-Verlag.
9. Michael Gelfond and Vladimir 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.

10. Michael Gelfond and Vladimir Lifschitz. Representing actions in extended logic programs. In K. Apt, editor, *Joint International Conference and Symposium on Logic Programming*, pages 579–573. MIT Press., 1992.
11. Juan Carlos Nieves, Mauricio Osorio, and Ulises Cortés. Preferred Extensions as Stable Models. *Theory and Practice of Logic Programming*, 8(4):527–543, July 2008.
12. Juan Carlos Nieves, Mauricio Osorio, and Claudia Zepeda. A schema for generating relevant logic programming semantics and its applications in argumentation theory. *Fundamenta Informaticae*, 106(2-4):295–319, 2011.
13. Mauricio Osorio, José R. Arrazola, and José Luis Carballido. Logical weak completions of paraconsistent logics. *J. Log. Comput.*, 18(6):913–940, 2008.
14. Mauricio Osorio, Juan Antonio Navarro, José Arrazola, and Verónica Borja. Logics with common weak completions. *Journal of Logic and Computation*, 16(6):867–890, 2006.
15. T. Cao Son and E. Pontelli. Planning with preferences using logic programming. In *Proceedings of the Logic Programming and Nonmonotonic Reasoning, 7th International Conference (LPNMR 2004)*, pages 247–260, Fort Lauderdale, FL, USA, 2004.
16. Dirk van Dalen. *Logic and Structure.* Springer, Berlin, second edition, 1980.