Mode-directed Preferences for Logic Programs

Hai-Feng Guo  
Department of Computer Science  
University of Nebraska at Omaha  
Omaha, NE 68182, USA  
haifengguo@mail.unomaha.edu

Bharat Jayaraman  
Dept. of Computer Science and Engineering  
State University of New York at Buffalo  
Buffalo, NY 14260, USA  
bharat@cse.buffalo.edu

ABSTRACT
Preference logic programming (PLP) is an extension of constraint logic programming for declaratively specifying problems requiring optimization or comparison and selection among alternative solutions to a query. PLP essentially separates the programming of a problem itself from the criteria specification of its solution selection. In this paper we provide a syntax for PLP based upon mode-directed preferences and a semantics based upon Herbrand models and fixed-point theory. Our method uses mode declarations to designate certain predicates as optimization predicates, and uses preference rules for stating the criteria for determining their optimal solutions. This paper also presents an elegant and easy method of executing preference logic programs in terms of tabled Prolog. Automatic transformation is applied to embed the preferences into the problem specification for efficient evaluation. We show that the procedural semantics of a preference logic program is equivalent to its declarative semantics.

Categories and Subject Descriptors
D.3.2 [Language Classifications]: Constraint and logic languages; F.3.2 [Semantics of Programming Languages]: Operational semantics

Keywords
Tabled resolution, Preference logic programming

1. INTRODUCTION
The paradigm of preference logic programming (PLP) [5, 6] was recently introduced as an extension of constraint logic programming (CLP) for declaratively specifying problems requiring optimization or comparison and selection among alternative solutions to a query. The PLP paradigm essentially separates the constraints of a problem itself from the criteria for selection the optimal solutions. The responsibility of how to find the optimal solution is shifted to the underlying logic programming system, in keeping with the spirit of logic programming as a declarative paradigm. Two important uses of PLP have been in the area of preference grammars [10] and preference queries in databases. More generally, the paradigm has been shown useful by practical applications in artificial intelligence, data mining, document processing, etc.

The PLP paradigm formulated in [5, 6] showed how the concept of preferences provides a natural, declarative, and efficient means of specifying a host of practical problems using definite clauses. The declarative semantics of PLP was given in terms of the 'preference consequences' of a program, i.e., truth in strongly optimal worlds [5, 6]. Reference [2] extended the work on preference logic grammars [10] to a three-valued preference logic grammars, and provided an implementation via normal logic programs. Other efforts such as [4, 11] incorporate optimization in a CLP framework, and [3] addresses semantics for optimization predicates in a CLP framework.

In this paper we provide a syntax for PLP based upon mode-directed preferences and a semantics based upon Herbrand models and fixed-point theory. The specification of the constraints of the general problem is separated from the specification of the criteria for the optimal solution. Their connection is established through a mode declaration scheme, which identifies certain predicates as ‘optimization’ predicates. Here, the term ‘constraints’ is used in a more general sense than the constraints of CLP, i.e., our ‘constraints’ are user-defined atoms that are defined using definite clauses. Therefore, the semantics of PLP can be defined as follows: the meaning of the constraints of the problem is defined in terms of its least Herbrand model [9]; and the meaning of preferences is defined as a strict partial order over the ground atoms of the optimization predicates. We characterize this semantics in terms of the least fixed-point of a transformation that is similar to the usual one for definite clauses [9].

This paper also presents an elegant method of specifying and executing preference logic programs in the paradigm of tabled Prolog. We focus on the definite logic programs in this paper. Preferences on alternative solutions to a query are defined using a set of well-defined logic clauses, which are separated from the logic programs to the problem specification. The connection between preferences and the problem specification is made through a formal mode declaration for tabled predicates. The execution of preference logic programs is achieved in two steps. First, a mode-directed automatic transformation is applied to embed the preferences...
into the problem specification for efficient evaluation. Second, the transformed program is then evaluated using tabled resolution, while the mode declaration provides a selection mechanism among the alternative solutions. We show that the procedural semantics of preference logic programs is consistent with its declarative semantics.

The rest of the paper is organized as follows: Section 2 gives a brief introduction on tabled Prolog systems, a mode declaration scheme for tabled predicates, and the syntax of mode-directed preferences. Section 3 presents the semantics of preference logic programs in terms of tabled Prolog programs, and how the mode declaration scheme in a tabled Prolog is extended for specifying and executing preference logic programs. Finally, section 4 gives our conclusions.

2. MODE-DIRECTED PREFERENCES

2.1 Mode Declaration

We use a tabled Prolog system [1, 7] to implement preference logic programs, because tabled Prolog can be thought of as an engine for efficiently computing fixed points. Consider the path program checking the existence of a path in Fig. 1(a). This program does not work in a traditional Prolog system. With the declaration of a tabled predicate path/2 in a tabled Prolog system, it can successfully find the complete solutions due to the fixed point computation strategy.

```
:- table path/2. :- table path/3.
path(X,Y) :- path(X,Y,[X,Y]).
edge(X,Y), path(X,Y). path(X,Y,[X,Y],L) :- edge(X,Y), path(X,Y,L).
path(X,Y) :- edge(X,Y). path(X,Y,[X,Y],L) :- edge(X,Y).
edge(a,b). edge(a,c). edge(a,b). edge(a,c).
edge(b,a). edge(b,a).
path(a,X). path(a,X,P).
```

(a) Finite Solutions (b) Infinite Solutions

![Figure 1: Tabled Prolog Programs](image)

The fixed point may sometimes contain an infinite number of solutions, which in turn affects the completion of the computation. Consider another path program shown in Fig. 1(b). An extra argument is added for the predicate path/3 to record the computed path. However, this argument causes an infinite number of paths to be enumerated logic programs, because tabled Prolog can be thought of as an engine for efficiently computing fixed points. The mode directive `table` can be further extended to associate a non-indexed argument of a tabled predicate with some optimum constraint. With the mode `last`, a non-indexed argument for each tabled answer only records the very first instance. This “very first” property can actually be generalized to support any other preferences, e.g., the minimum/maximum value with mode `min`/`max`, etc. Contrary to the mode `last`, the mode `last` is useful for recording the last answer from a solution set. This mode can be realized as follows: for a tabled call, whenever a new answer is generated, it will replace the old tabled one if any so that the tabled answer is always the ‘last’ answer generated so far. The mode `<<` is used to support user-defined preferences. The incorporating uses of the modes `last` and `<<` provides an elegant interface for preference logic programming.

2.2 Syntax

In general there are two components to an optimization problem: specification of the constraints of a problem, and specification of what and how to be optimized. The intent of preference logic programming is to separate these two components and declaratively specify such applications.

**Definition 1 (Preference Logic Programs).**

A (definite) preference logic program P can be defined as a pair $<$pref, Pcore$, where Pcore and Pref are two disjoint sets of clauses defined as follows: Pcore specifies the constraints of the problem as a set of definite clauses; Pref defines the optimization criteria using a set of preference clauses (or preferences) of the form:

$$p(T_1) <<< p(T_2) : B_1, B_2, ..., B_n. \quad (n \geq 0)$$

where $K(p(T_1)) = K(p(T_2))$ and each $B_i$ ($1 \leq i \leq n$) is an atom defined in Pcore.

The informal semantics of $p(T_1) <<< p(T_2) : B_1, B_2, ..., B_n$ is that the atom $p(T_1)$ is less preferred than $p(T_2)$ if $B_1, B_2, ..., B_n$ are all true. Note that the two atoms being compared have the same predicate symbol $p$. Also, $K(p(T_1)) = K(p(T_2))$ states that only two atoms with the same corresponding indexed arguments (mode `+`) are comparable.

**Example 1.** Consider the following preference program searching for a shortest path, where `path(X,Y,D,L)` denotes a path from `X` to `Y` with the distance `D` and the path route `L`.

<table>
<thead>
<tr>
<th>Modes</th>
<th>Informal Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>an indexed argument</td>
</tr>
<tr>
<td>$-$</td>
<td>a non-indexed argument</td>
</tr>
<tr>
<td>$\textit{last}$</td>
<td>a non-indexed argument for the last answer</td>
</tr>
<tr>
<td>$\textless \textless$</td>
<td>a user-defined preference mode</td>
</tr>
</tbody>
</table>

**Table 1: Built-in Modes for Tabled Predicates**
Clauses (2) to (5) make up the core program $P_{\text{core}}$ defining the path relation and a directed graph with a set of edges; clause (1) and (6), the preference clauses $P_{\text{pref}}$, specifies the predicate `path/4` to be optimized and gives the criteria how to optimize the `path/4` predicate, that is, the path for each pair of reachable nodes (according to the first two indexed arguments in `path/4`) should be optimized based on the definition of `<`: the shorter path is preferred.

3. SEMANTICS OF PREFERENCES

3.1 Fixed Point Semantics

The declarative semantics of a preference program is based on the Herbrand models and fixed-point theory [9]. The preference specifications induce a partial order over ground atoms and can be characterized as the least fixed point of preference specifications. We use the following atoms and can be characterized as the least fixed point of preference logic programs:

- $\forall \pi \in I$.

Note that the model for $P_{\text{core}}$ follows the standard model definition [9] for definite clauses, which is different from Def. 3 for a preference program $P$. Def. 3(c) tells that no better-preferred atom $A_1$ than the optimized atom $A$ can be found in the least Herbrand model $P_{\text{core}}$, otherwise, $A$ cannot be an optimized atom. However, there may exist a Herbrand atom $A_1 \notin P_{\text{core}}$ better-preferred than $A$.

We wish to obtain the link between the models of $P$ and $P_{\text{core}}$ so that we can find out how preferences affect the semantics of a general program. For this we need to introduce two new meta-level mappings defined over Herbrand interpretations.

**Definition 4.** Let $P$ be a preference program, $M$ be a Herbrand model for $P_{\text{core}}$, and $M_1$ be a subset of $M$ containing all the atoms of any optimization predicate. We define a meta-level mapping $\phi: M \rightarrow 2^{M_{P_{\text{core}}}}$ as follows:

$$\phi_P(M) = \{A \in M_1 : \exists A_1 \in M_1 \text{ s.t. } A \prec A_1\}.$$

**Definition 5.** Let $P$ be a preference program. We define a meta-level mapping $\phi: M \rightarrow 2^{M_{P_{\text{core}}}}$ as follows:

$$\phi_P(M) = \{A \in M : A \prec A_1, \cdots, A_n \text{ is a ground instance of a clause in } P_{\text{core}} \text{ and } \{A_1, \cdots, A_n\} \subseteq M\}.$$

The above two mappings provide the link between the declarative and procedural semantics of a preference program. The mapping $\phi_P$ filters suboptimal atoms from the model according to the preference relation; the mapping $\phi_P$ filters those atoms depending on the removed suboptimal atoms from the model. It is obvious that $\phi_P(I) \subseteq I$ for any given Herbrand interpretation $I$. Thus, we come to a major result of the theory as shown in the next theorem.

**Theorem 1.** Let $P$ be a preference program and $M_{P_{\text{core}}}$ be the least Herbrand model for $P_{\text{core}}$. Then $M_P = \phi_P \uparrow \omega(\phi_P(M_{\text{core}}))$ is an intended model for $P$.

**Proof:** We show how $M_P$ satisfies the properties (a), (b), and (c) as defined in the Def. 3 for an intended model:

1. Based on the definition of $\phi_P$, it is clear that $\phi_P(M_{\text{core}})$ satisfies the property (b): Since $\pi_P(I) \subseteq I$ for any given Herbrand interpretation $I$, $\pi_P \uparrow \omega(\phi_P(M_{\text{core}}))$ satisfies the property (b) too.

2. We associate a complete lattice with the program $P$, $2^{M_{\text{core}}}$, the set of all Herbrand interpretations of $P_{\text{core}}$ and $P$, is a complete lattice under the partial order of set inclusion $\subseteq$, where the top element is $B_P$ and the bottom element is $\emptyset$. Thus, $M_P = \pi_P \uparrow \omega(\phi_P(M_{\text{core}}))$ must be a fixed point.
of \( \pi_P \) over the lattice, that is, \( \pi_P(M_P) = M_P \). Therefore, \( M_P \) satisfies the property (a), and hence it is a model for \( P \).

3. Let \( q/n \) be an optimization predicate and \( A \) be one atom of \( q/n \) in \( M_P \). Assume that there exists a Herbrand atom \( A_1 \in M_F \) s.t. \( A \prec A_1 \). According to the definition of \( \phi_P \) in Def. 4, \( A \notin \phi_P(M_{core}^P) \), and hence \( A \notin M_P \), which is a contradiction to the fact that \( A \in M_P \). Therefore, \( M_P \) satisfies the property (c).

Thus, \( M_P \) is an intended model for \( P \). □

If we reconsider the preference program in the Example 1. Its least Herbrand model \( M_{core}^P \) and \( \phi_P(M_{core}^P) \) are shown below:

\[
M_{core}^P = \{ \\
\quad \text{edge}(a, b, 4), \text{edge}(b, a, 3), \text{edge}(b, c, 2), \\
\quad \text{path}(a, a, 0, \{ \}), \text{path}(a, a, 7, \{a, b\}, \{b, a\}), \ldots \\
\quad \text{path}(a, b, 4, \{a, b\}), \text{path}(a, b, 11, \{a, b\}, \{b, a\}), \ldots \\
\quad \ldots \\
\quad \text{path}(c, c, 0, \{ \}) \} \\
\phi_P(M_{core}^P) = \{ \\
\quad \text{edge}(a, b, 4), \text{edge}(b, a, 3), \text{edge}(b, c, 2), \\
\quad \text{path}(a, a, 0, \{ \}), \text{path}(a, a, 7, \{a, b\}, \{b, a\}), \ldots \\
\quad \text{path}(a, b, 4, \{a, b\}), \text{path}(a, b, 11, \{a, b\}, \{b, a\}), \ldots \\
\quad \ldots \\
\quad \text{path}(c, c, 0, \{ \}) \}
\]

We also have \( \pi_P \uparrow \omega(\phi_P(M_{core}^P)) = \phi_P(M_{core}^P) \) for this program. However, if we add an extra clause

\[
\text{shortest}(X, Y, D, P) :- \text{path}(X, Y, D, P) .
\]

then \( \pi_P \uparrow \omega(\phi_P(M_{core}^P)) \) is different from \( \phi_P(M_{core}^P) \). e.g.,

\[
\text{shortest}(a, a, 7, \{a, b\}, \{b, a\}) \in \phi_P(M_{core}^P), \text{but} \\
\text{shortest}(a, a, 7, \{a, b\}, \{b, a\}) \notin \pi_P \uparrow \omega(\phi_P(M_{core}^P)).
\]

**Corollary 2.** Let \( P \) be a preference program and \( q/n \) be an optimization predicate. \( A \) is an atom of \( q/n \) and \( A \in M_P \) if and only if \( A \) is an optimized atom in \( M_{core}^P \).

**Proof:** It is shown based on the Def. 3(c) since \( M_P \) is an intended model for \( P \). □

### 3.2 Procedural Semantics

A preference program \( P \) can be automatically transformed to a new tabled program by incorporating the general problem specification \( P_{core} \) and the optimization criteria \( P_{pref} \). The optimization criteria are then elegantly embedded to filter the suboptimal answers. Thus, the procedural semantics of a preference logic program is dependent on that of a tabled program.

It is worthwhile to be mentioned that a preference program may contain contradictory preferences (e.g., \( p(a) \ll \ll p(b) \) and \( p(b) \ll \ll p(a) \)), whose preference relation does not make sense in practice. In the general case, the detection of contradictory preferences is an undecidable problem; even for the programs with bound-size terms (i.e., finite domain), the detection is still an NP-hard problem. Therefore, we only focus on those preferences whose relations are strict partial order\(^1\).

**Example 2.** Consider the following transformed tabled program from the program in Example 1.

\[
\begin{align*}
\text{pathNew}(X, X, O, [ ]) & :- \text{edge}(X, Y, D). \\
\text{pathNew}(X, Y, D, [X, Y]) & :- \text{edge}(X, Y, D). \\
\text{pathNew}(X, Y, D, [X, Z] \mid P) & :- \text{edge}(X, Z, D_1), \text{path}(Z, Y, D_2, P), D = D_1 + D_2.
\end{align*}
\]

\(^1\)A strict partial order relation is irreflexive and transitive.

\[
\begin{align*}
\text{prefer}(T1, T2) & :- T1 \ll \ll T2. \\
\text{prefer}(T1, T2) & :- T1 \ll \ll T3, \text{prefer}(T3, T2). \\
\text{path}(X, Y, D_1, _) & \ll \ll \text{path}(X, Y, D_2, _). \\
\text{path}(X, Y, D, E) & :- \\
& \text{path}(X, Y, D, E), \\
& \text{path}(X, Y, D_1, E), \\
& \text{path}(X, Y, D, E) \\
\end{align*}
\]

Four major changes have been made in this transformation by taking advantage of the unique global table in the system:

(i) A new predicate \( \text{prefer}/2 \) is introduced to define the corresponding preference relation \( \prec \), such that \( \text{prefer}(T1, T2) \) if and only if \( T1 \prec T2 \).

(ii) The original predicate \( \text{path}/4 \) in Example 1 is replaced by a new predicate \( \text{pathNew}/4 \) to emphasize that this predicate is to generate a new preferred path candidate from \( X \) to \( Y \).

(iii) The predicate \( \text{path}/4 \), given a new definition as the clause (9), represents the way how to identify an preferred answer. The meaning of the clause (10) is the following: a path candidate \( A \) by \( \text{pathNew}(X, Y, D, E) \), we check whether there already exists a tabled answer, and, if so, they are compared to each other to keep the preferred one in the table; otherwise, the candidate is recorded as a first tabled answer. By this way, the solution(s) eventually left in the table must be the optimal one(s).

(iv) The table mode \( \ll \ll \) for \( \text{path}/4 \) is hence replaced by \( \text{last} \) to catch the optimal tabled answer.

**Definition 6 (\( \rho \)-Transformation).** Let \( P \) be a preference program. The transformation to a new tabled program \( P' = \rho(P) \) can be formalized as follows: for any optimization predicate \( q/n \) in \( P \), we have

- a new predicate \( \text{prefer}/2 \) defined as:

\[
\text{prefer}(T1, T2) :- T1 \ll \ll T2. \\
\text{prefer}(T1, T2) :- T1 \ll \ll T3, \text{prefer}(T3, T2).
\]

- for each clause defining \( q/n \) in \( P \), the predicate \( q/n \) in its head (the part to the left of :-) is renamed to a new predicate \( q/n' \);

- a new clause definition for \( q/n \) is introduced in the form of:

\[
q(a_1, \ldots, a_n) :- q'(a_1, \ldots, a_n), \\
q(b_1, \ldots, b_n) \\
\rightarrow \text{prefer}(q(b_1, \ldots, b_n), q(a_1, \ldots, a_n)) ; \text{true}.
\]

where \( a_1, \ldots, a_n \) and \( b_1, \ldots, b_n \) are all variables; for \( 1 \leq i, j \leq n, a_i \) and \( b_i \) use same variable if \( i = j \); \( a_i \) and \( a_j \) (or \( b_i \) and \( b_j \)) use two different variables if \( i \neq j \);

- in the mode declaration of \( q/n \), the first preference mode \( \ll \ll \) is changed into \( \text{last} \), and the rest preference modes (if any) are changed into the standard non-indexed mode \( \sim \);

- for the other clauses not defining optimization predicates in \( P \), make a same copy into \( P' \).
We call this transformation $\rho$-transformation.

The procedural semantics of a tabled program is dependent on tabled resolution [1, 7]. In spite of having different tabled resolution, a tabled Prolog can be thought of as an engine for efficiently computing the least fixed points. The procedure of computing fixed points of a definite tabled logic program mimics the bottom-up computation strategy [7].

**Definition 7.** Let $P$ and $B_P$ be a program and its Herbrand base. We define a meta-level procedure $T_P : 2^{B_P} \rightarrow 2^{B_P}$. Given a Herbrand interpretation $I$, $T_P(I)$ performs:
1. $I_0 = \emptyset$;
2. for each ground instance $A : A_1, \ldots, A_m$ of a clause in $P$ where $\{A_1, \ldots, A_m\} \subseteq I$, do
   - if $A$ is an atom of a tabled predicate with mode 'last':
     - $I_0 \leftarrow (I_0 \cup \{A\}) - \{A_1 \in I : K(A_1) = K(A)\}$ ($\ast$)
   - else
     - $I_0 \leftarrow I_0 \cup \{A\}$;
3. return $I_0$.

Thus, the fixed point semantics of $P$ can be described as $T_P \uparrow \omega(\emptyset)$.

Def. 7 gives the procedural semantics for a transformed program. The statement ($\ast$) shows the procedural semantics of the mode 'last'. Thus, we have the following major results. Theorem 4 shows the equivalence between the declarative semantics of a preference program and its procedural semantics over a transformed tabled program.

**Proposition 3.** Let $P$ be a preference program and $q/n$ be an optimization predicate. $A$ is an atom of $q/n$ and $A \in T_{\rho(P)} \uparrow \omega(\emptyset)$ if and only if $A$ is an optimized atom in $M_{\text{core}}^P$.

**Proof:** It is shown based on the transformed definition of $q/n$ in Def. 6. $\square$

**Theorem 4.** Let $P$ be a preference program. Then $\pi_P \uparrow \omega(\phi(P(M_{\text{core}}^P))) = T_{\rho(P)} \uparrow \omega(\emptyset)$, where $M_{\text{core}}^P = T_{\rho core} \uparrow \omega(\emptyset)$.

**Proof:** Let $A$ be a Herbrand atom. The proof is based on the following two cases:
(i) If $A$ is an atom of an optimization predicate, $A \in \pi_P \uparrow \omega(\phi(P(M_{\text{core}}^P)))$ $\Leftrightarrow A$ is an optimized atom in $M_{\text{core}}^P$. Corollary 2 $\Leftrightarrow A \in T_{\rho(P)} \uparrow \omega(\emptyset)$ Proposition 3
(ii) If $A$ is not an atom of an optimization predicate, the proof can be done by a structural induction on the definition of the predicate for $A$ (omitted due to the page limit). $\square$

4. **CONCLUSIONS**

The underlying philosophy of this paper may be expressed by the equation: Programming = Logic + Preferences + Control. This paradigm is particularly suited to those problems requiring optimization or comparison and selection among alternative solutions. It allows logic (constraints of the problem) and preferences (the criteria for the optimal solutions) to be specified separately in a declarative fashion.

The declarative semantics of a preference logic program is based upon Herbrand models. Preference specifications essentially induce a partial order on ground atoms, and the intended model is defined in terms of the most preferred atoms according to this order. We show that this intended model can be characterized as the least fixed point of a natural meta-level mapping operation over the least Herbrand model of the core program.

This paper presents an elegant method of specifying and executing preference logic programs in the paradigm of tabled Prolog. A preference logic program $P$ can be automatically transformed to a new tabled program by incorporating the general problem specification $P_{\text{core}}$ and the optimization criteria $P_{\text{pref}}$. The optimization criteria are elegantly embedded to filter the suboptimal answers. Therefore, the procedural semantics of a preference logic program is reduced to that of a tabled logic program, which is equivalent to computing its least fixed point via a bottom-up computation strategy. We have also briefly shown that the declarative semantics of a preference logic program is equivalent to its procedural semantics via the transformation to a tabled program.

**Acknowledgments**

This research was partially supported by the Nebraska NSF EPSCOR grant.

5. REFERENCES