Extensions of Logic Programming for Preference Representation

Antonis Troumpoukis*

National and Kapodistrian University of Athens,
Department of Informatics and Telecommunications
antru@di.uoa.gr

Abstract. We consider the problem of preference representation using extensions of logic programming. In this dissertation, we propose two approaches for expressing preferences. Regarding the first approach, we propose PrefLog, a logic programming language which uses an underlying infinite-valued truth domain in order to support quantitative preference operators. We introduce the syntax and the semantics of the language, and we study the properties of the PrefLog operators that are needed in order for programs to behave well from a semantic point of view. In addition, we introduce a terminating bottom-up evaluation method for a well-defined class of function-free PrefLog programs. Regarding the second approach, we propose the use of higher-order logic programming as a framework for representing qualitative preferences. In this approach, relations, preferences between tuples, preferences between sets of tuples and operations on preferences are expressed in the same, higher-order language. The programs can be evaluated by standard higher-order programming systems, and their performance can be enhanced with generic and specialized optimization techniques. Among these techniques, we propose a novel program transformation technique for translating higher-order programs into first-order ones and we use this technique for optimizing the higher-order programs of our interest.

1 Introduction

Preferences play a major role in human life. They can affect us in many situations; from simple personal choices in early childhood (e.g., "which ice-cream flavor do you prefer?") up to complex professional decisions (e.g., "should I pursue a career in music or in computer science?"). Therefore, it comes as no surprise that preferences have been explored in many scientific disciplines (such as Philosophy, Economics, and Psychology). Research in preferences is very active in Computer Science, mostly in areas such as Artificial Intelligence [10], Database Systems [20], and Programming Languages [9]. One of the main objectives of the study of preferences in Computer Science is the design of languages and frameworks that can provide us with the ability to choose among alternatives in a declarative way, whether these alternatives are problem solutions, program

* Dissertation Advisor: Panos Rondogiannis, Professor

answers, or query results. Effective user preference representation formalisms can be applied in information systems so that the responses presented to the users can be more compact and comprehensive because it can reflect their true interests.

Preference representation formalisms usually fall into two basic categories [20]. In the *quantitative* approach [1, 2, 3, 15], preferences are represented by a preference value function. Each object is associated with a *preference score*, which is a numerical value that expresses the degree of interest (e.g., "my preference in beer is 0.9 while in wine it is 0.2"). In the *qualitative* approach [7, 8, 13, 14, 25] preferences are expressed by direct comparisons between objects (e.g., "I prefer beer over wine"), thus resulting in a *binary preference relation*. Each category has its strengths and its weaknesses. The qualitative approach is more general than the quantitative approach (i.e., not all preference relations can be expressed by scoring functions or through degrees of interest [7, 20]). On the other hand, quantitative preferences can distinguish how much preferred one object is over another (e.g., a preference score of 0.9 is much more preferred than a score of 0.001, but is a little more preferred than a score of 0.899).

As a general observation, we could say that both qualitative and quantitative formalisms that have been developed leave much room for improvement both in terms of expressiveness and efficiency. Quantitative approaches usually rely on the definition of a preference function. However, a preference function cannot always be defined, and if it can, users in most cases are rarely willing to express their preferences directly in terms of such a function. Qualitative approaches have their weaknesses too; first, they usually offer a quite limited set of preference operations—in most cases, only one preference operator can be used (namely, "find the most preferred objects according to this preference"); second, almost all approaches use two distinct languages, one for representing the base knowledge and one for representing the preferences, making the structure of the representation non-uniform. Moreover, qualitative approaches rely on structures such as *partial order relations*, which are more complex than simple numerical values; therefore, the process of handling a qualitative preference is clearly a more complex task than that of a quantitative preference. As a result, the development of optimization techniques for enhancing the performance of qualitative frameworks can be of crucial importance from a practical point of view.

The purpose of this dissertation is to study new, more expressive formalisms for representing and manipulating preferences. In particular, we use two extensions of logic programming:

- The first approach uses infinite-valued logic programming for expressing quantitative preferences. This language is based on an infinite set of truth values in order to support operators for expressing preferences.
- The second approach uses higher-order logic programming for expressing qualitative preferences. In this approach, preference relations and operations on preferences are expressed in the same, higher-order language.

Our approaches attempt to overcome the shortcomings that were mentioned previously. In our quantitative approach, the preference values are not denoted

directly but are expressed using appropriate preference operators. In our qualitative approach, base relations, preferences, and operations on preferences are represented using the same language making our approach more uniform. In addition, our framework allows the definition of many preference operators other than those that are offered in most qualitative approaches in the literature.

Our contributions can be summarized as follows:

- We argue that the adoption of many-valued logics is a promising idea for developing expressive new preferential logic programming languages. We define the simple preferential logic programming language PrefLog which differs from other preferential logic programming approaches in that it uses an underlying infinite-valued truth domain in order to support quantitative preference operators. We show that the continuity of these operators over the infinite-valued underlying domain ensures that the resulting logic programming system retains all the standard and well-known properties of classical logic programming (and most notably the existence of a least Herbrand model).
- We demonstrate that terminating bottom-up evaluation can be performed for a large function-free fragment of PrefLog. This result is not obvious: despite the fact that the Herbrand Base of the programs we consider is finite, an atom may obtain an infinity of truth values during a bottom-up evaluation, resulting in possible non-termination.
- We argue that higher-order logic programming is a very expressive framework for representing and manipulating qualitative preferences. A significant advantage of our approach is that preference formulas as-well-as operators that are parameterized with such formulas can be expressed in the same language. Moreover, the seemingly more demanding case of preferences over sets can be handled without extra notational overhead, because preferences over sets are essentially second-order relations and can, therefore, be encoded easily in our higher-order language.
- We implement specialized techniques that can enhance higher-order logic programming so that it can better handle and manipulate preferences. We propose Predicate Specialization, a transformation technique based on the abstract framework of *Partial Evaluation*. This technique is used for optimizing higher-order logic programs that express preferences over tuples by transforming them into first-order ones. Moreover, we implement two custom-tailored implementation strategies for optimizing set-preference higher-order programs. Finally, we provide experimental results that suggest that the proposed techniques can enhance the performance of our higher-order framework.

The rest of this document is organized as follows: in Section 2, we present PrefLog, an extension of classical logic programming for expressing quantitative preferences; in Section 3 we present a higher-order logic programming framework for expressing quantitative preferences; and finally, we close with conlusions and some pointers to future work.

2 Quantitative Preferences and Infinite-Valued Logic Programming

2.1 An intuitive overview

The central idea of the infinite-valued approach can be summarized as follows: *We can represent quantitative preferences with an infinite set of truth values, such that the different levels of truth correspond to different degrees of preference. In order for the description to be more natural though, the manipulation of the different levels of preference should not be done by processing the truth values directly, but with the use of operators that resemble preference operations that appear in natural languages.* The above idea can be illustrated in the following example:

Example 1. Suppose that we are using an online flight reservation system in order to fly from Athens to Boston. Assume that we want to book a flight ticket from Athens to Boston, *and optionally* flying with "Reliable Airlines". This fact can be encoded as follows:

$$\texttt{desired\_flight(F)} \leftarrow \texttt{from\_to(athens, boston, F)} \wedge \texttt{opt carrier(F, reliable\_air)}.$$

The above program looks like a classic logic program, with the difference that it uses the opt operator. If this operator is not present then the atom expresses a necessary condition; otherwise, the atom expresses an optional condition that it "would be preferable but not necessary, to be satisfied". Now, suppose that we issue a query of the form:

$$\leftarrow \texttt{desired\_flight(F)}.$$

If the query succeeds then we found a flight from Athens to Boston with Reliable Airlines. If the query completely fails then there does not exist any flights from Athens to Boston. If the query partially fails (meaning that it is evaluated into an intermediate truth value), then a flight has been found which however is not with Reliable Airlines. This means that we can fly to our destination, but not traveling with the carrier of our preference.

In order to formulate this approach [17, 18] we introduce the logic programming language PrefLog, its syntax, and its semantics; in particular, we study the properties of the PrefLog operators that are needed in order for the PrefLog programs to behave well from a semantic point of view. In addition, we introduce a bottom-up evaluation method for a well-defined class of function-free PrefLog programs.

2.2 The logic programming language PrefLog

We extend classical logic programming with a set of truth values $\mathbb{V}$ with the following ordering:

$$F_0 < F_1 < F_2 < \cdots < 0 < \cdots < T_2 < T_1 < T_0$$

Apart from the standard true value, denoted by T_0, this set also contains the truth values $T_1, T_2, \ldots$ that are less and less "true" than T_0; moreover, apart from the standard false value, denoted by F_0, it also contains the values $F_1, F_2, \ldots$ that are less and less "false" than F_0. This set of truth values was originally proposed by Rondogiannis and Wadge in order to provide a purely model-theoretic semantics for logic programming with negation-as-failure [19].

In PrefLog, the preferences are not denoted directly with elements from $\mathbb{V}$, but are expressed using appropriate preference operators that resemble preference operations that appear in natural languages. More formally, the "semantic meaning" of a n-ary operator ∇/n, is a function $\|\nabla\| : \mathbb{V}^n \to \mathbb{V}$. Some examples of preference operators are the following:

$$\|\wedge\| = \min \qquad\qquad \|\vee\| = \max$$

$$\|\mathsf{opt}^k\|(v) = \begin{cases} F_{i+k}, & v = F_i \\ 0, & v = 0 \\ T_i, & v = T_i \end{cases} \qquad \|\mathsf{alt}^k\|(v) = \begin{cases} F_i, & v = F_i \\ 0, & v = 0 \\ T_{i+k}, & v = T_i \end{cases}$$

Apart from these operators, one could define more preference operators.

In the remaining part of this subsection we will define the semantics of PrefLog. In particular, we study the properties of the PrefLog operators that are needed in order for programs to behave well from a semantic point of view.

Definition 1. *Let $\mathbf{x}, \mathbf{y} \in \mathbb{V}^n$, $n \geq 1$ where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$. We write $\mathbf{x} \leq \mathbf{y}$ if $x_i \leq y_i$, for all $1 \leq i \leq n$.*

Proposition 1. *Let $n \geq 1$. Then, $(\mathbb{V}^n, \leq)$ is a complete lattice.*

Definition 2. *Let ∇/n be a preference operator. Then, ∇ is called* continuous *if for all sequences $(\mathbf{x}_n)_{n\geq 0}$ of elements of $\mathbb{V}^n$ such that for all $n \geq 0$, $\mathbf{x}_n \leq \mathbf{x}_{n+1}$ it holds $\|\nabla\|(\mathsf{lub}(\{\mathbf{x}_n : n \geq 0\})) = \mathsf{lub}\{\|\nabla\|(\mathbf{x}_n) : \mathbf{x}_n \geq 0\}$.*

Theorem 1. *Let P be a PrefLog program. If all operators used in the bodies of rules of P are continuous, then P has a least (with respect to $\leq$) model M_{P}.*

The previous theorem suggests that any preference operator can be used by PrefLog programs as long as it is continuous. In the case of the preference operators illustrated previously, we have the following proposition:

Proposition 2. *The operators $\wedge, \vee, \mathsf{opt}^n, \mathsf{alt}^n$ are continuous.*

2.3 Bottom-up evaluation of a function-free fragment of PrefLog

We reduce our focus in $\{\epsilon, \wedge\}$-programs, which are PrefLog programs that do not use functional symbols and contain only the operators $\wedge$ and ϵ, where:

$$\|\epsilon\|(v) = \begin{cases} F_{i+1}, & v = F_i \\ 0, & v = 0 \\ T_{i+1}, & v = T_i \end{cases}$$

The operator ϵ is continuous, therefore is a valid PrefLog operator. Even though it can be shown that there exist valid preference operators that cannot be defined using $\wedge$, $\vee$ and ϵ, all programs that we have used in the dissertation are constructed using these operators. For instance, for opt^n and alt^n it holds:

$$\mathsf{opt}^n\ \mathsf{A} \equiv \mathsf{A} \vee \overbrace{\epsilon\ \epsilon\ \cdots\ \epsilon}^{n}\ \mathsf{A} \qquad\qquad \mathsf{alt}^n\ \ \mathsf{A} \equiv \mathsf{A} \wedge \overbrace{\epsilon\ \epsilon\ \cdots\ \epsilon}^{n}\ \mathsf{A}$$

This fact suggests that every program that uses $\wedge, \vee, \mathsf{opt}^n, \mathsf{alt}^n$ or any other operator that can be defined using these operators can be transformed into an equivalent $\{\epsilon, \wedge\}$-program.

The most direct way of evaluating the minimum model of a $\{\epsilon, \wedge\}$-program P in a bottom-up manner is to begin by assigning F_0 to all ground atoms of P, use the rules of P to compute new values for all atoms, and repeat this procedure until the values of all atoms stabilize. Even though this procedure terminates for classical Datalog programs, this does not hold in our case, because each atom can receive any truth value from the infinite set $\mathbb{V}$. For example, the program

$$\mathtt{p} \ \leftarrow\ \epsilon\, \mathtt{p}.$$

has the minimum-model $\{(\mathtt{p}, 0)\}$ which is produced using an infinite number of steps, namely: $\{(\mathtt{p}, F_0)\} \rightsquigarrow \{(\mathtt{p}, F_1)\} \rightsquigarrow \{(\mathtt{p}, F_2)\} \rightsquigarrow \cdots \rightsquigarrow \{(\mathtt{p}, 0)\}$.

A solution is to evaluate the minimum model *in stages*, using the followng *Terminating Bottom-up Evaluation* strategy:

1. Set F_0 to all atoms.
2. For $k = 0, 1, \ldots$
   (a) Use the rules of the program to compute new values for all atoms.
   (b) Repeat until the set of atoms that have F_k, T_k values stabilize.
   (c) If there are no remaining atoms (i.e., atoms with value with order $> k$) :
      i. End the evaluation.
   (d) If a *gap* (i.e., no atoms with value with order k) is produced:
      i. Set 0 to all remaining atoms.
      ii. End the evaluation.
   (e) Set F_{k+1} to all remaining atoms.

For example, the minimum model of the program

$$\begin{array}{l} \mathtt{p} \ \leftarrow\ \epsilon\, \mathtt{p}. \\ \mathtt{q} \ \leftarrow\ \mathtt{r}. \\ \mathtt{r}. \end{array}$$

is evaluated using the above strategy as follows: $\{(\mathtt{p}, F_0), (\mathtt{q}, F_0), (\mathtt{r}, F_0)\} \rightsquigarrow \{(\mathtt{p}, F_1), (\mathtt{q}, F_0), (\mathtt{r}, T_0)\} \rightsquigarrow \{(\mathtt{p}, F_2), (\mathtt{q}, T_0), (\mathtt{r}, T_0)\} \rightsquigarrow \{(\mathtt{p}, F_3), (\mathtt{q}, T_0), (\mathtt{r}, T_0)\} \rightsquigarrow$ all atoms with values F_0, T_0 stabilize $\rightsquigarrow \{(\mathtt{p}, F_1), (\mathtt{q}, T_0), (\mathtt{r}, T_0)\} \rightsquigarrow \{(\mathtt{p}, F_2), (\mathtt{q}, T_0), (\mathtt{r}, T_0)\} \rightsquigarrow$ gap at order 1 $\rightsquigarrow \{(\mathtt{p}, 0), (\mathtt{q}, T_0), (\mathtt{r}, T_0)\}$.

For this evaluation algorithm we have the following theorem. Regarding the correctness proof, we use material from [11, 19]. Regarding the termination proof, we use the fact that $\{\epsilon, \wedge\}$-programs do not have gaps in their minimum models.

Theorem 2. *The Terminating Bottom-up Evaluation correctly computes the least model M_{P} of any given $\{\epsilon, \wedge\}$-program P in a finite number of steps.*

3 Qualitative Preferences and Higher-Order Logic Programming

3.1 Background

The starting point of our approach is an influential proposal by J. Chomicki [7] for representing qualitative preferences in the context of relational database systems. Chomicki's approach is based on the following two ideas:

- Preferences between tuples of a database relation are specified using binary *preference relations*; these relations are defined using first-order formulas, called *preference formulas*.
- A new relational algebra operator is introduced. This operator is called `winnow` and takes two parameters; a database relation and a preference formula. The `winnow` operator selects from its input relation the most preferred tuples according to the given preference formula. In particular:

$$\texttt{winnow}_C(R) = \{t \in R : \neg\exists t' \in R \text{ s.t. } t' \succ_C t\}$$

Example 2. Suppose that we have a relation of movies. Now, suppose that we want to express the following preference relation: *"Prefer one movie over another iff their genres are the same and the rating of the first is higher"*. This preference relation can be described using the following preference formula:

$$t_1 \succ_C t_2 \equiv (t_1.\mathsf{genre} = t_2.\mathsf{genre}) \wedge (t_1.\mathsf{rating} > t_2.\mathsf{rating})$$

As a result, the query $\texttt{winnow}_C(\texttt{movie})$ will return the most preferred movies from the relation `movie` using the preference relation C of our interest.

The approach advocated by Chomicki, despite groundbreaking, has certain limitations. First, it supports only *intrinsic* preferences (that is preference relations that depend solely on the basis of the values occurring in the tuples). Second, the preference relations and the preference queries are expressed in two different languages, namely, first-order logic and SQL extended with the `winnow` operator, which makes the approach less uniform. Third, there is no way to define *directly* other operators apart from `winnow`. And finally, the framework cannot be extended in a straightforward and elegant way for expressing preferences over sets (consider the extension of Zhang and Chomicki [25] which uses additional concepts such as *features* and *profiles*).

3.2 Expressing Preferences using Higher-Order Logic Programming

The central idea of the higher-order approach can be summarized as follows: *Since qualitative preferences can be expressed using binary preference relations, and since operations on preferences involve operations that take preference relations as arguments, a higher-order language can offer increased representation capabilities. Moreover, sets of tuples are also relations, therefore preferences over sets are essentially second-order relations and can encoded easily in a higher-order language.* The above idea can be illustrated in the following examples:

Example 3. Suppose that we have a relation of movies encoded with facts of the form `movie((Name,Genre,Rating))`. The preference relation of Example 2 can be encoded easily using the following logic program:

```
c_pref((N1,G,R1), (N2,G,R2)) :- movie((N1,G,R1)),
                                movie((N2,G,R2)), R1 > R2.
```

The above program is a first-order one, so it does not use any higher-order features. However, if we want to define an operator that processes preference relations such as the above, we need to define higher-order predicates. For example, the following operator `winnow(C,R,T)` returns the best tuples `T` from a relation `R` according to a binary preference relation `C`:

```
winnow(C,R,T) :- R(T), not bypassed(C,R,T).
bypassed(C,R,T) :- R(Z), C(Z,T).
```

Notice that the above program is a higher-order one, since it contains variables that appear in places where predicates typically occur, and predicates that can accept other predicates as arguments. As a result, the following query:

```
?- winnow(c_pref,movie,T).
```

will return the most preferred movies from the relation `movie` using the preference relation `c_pref` of our interest.

Example 4. Suppose that we have a relation of movies and we want to express the following preference between sets of movies: *"we want to watch 3 movies and we prefer the sum of the ratings of the movies to be the highest possible"*. In the following higher-order program which encodes the above preference, we assume the existence of a predicate `rating_sum(S,N)`, which returns in the variable `N` the sum of the ratings of the elements of the set `S`:

```
rating_pref(S1,S2) :- rating_sum(S1,N1),
                      rating_sum(S2,N2), N1 > N2.
```

Moreover, assume the existence of a predicate `subsetof(R,N)(S)`, which returns in the variable `S` all subsets of size `N` from the relation `S`. This syntax with the extra pairs of parentheses allows us to use the feature of *partial applications*, i.e., the ability to invoke a higher-order predicate with only some of its arguments. For instance, the expression `subsetof(movie,3)` represents the relation of all subsets of `movie` of size 3. As a result, the following query:

```
?- winnow(rating_pref,subsetof(movie,3),S).
```

will return the most preferred sets of movies accding to the preference relation `rating_pref` of our interest. Notice that the `winnow` operator is the same as previously, but now it is third order. We note that the actual implementation of the `subsetof` predicate depends on the higher-order language that we are using, because not all higher-order logic programming languages treat existential predicate variables with the same manner.

Example 5. The higher-order features allow us to define generic operators on preference relations. For example, consider a relation `r` and two preference relations `c1_pref` and `c2_pref`. Suppose now that we have the following preference that combines these preference relations: *"Prefer a tuple t_1 from a tuple t_2 using* `c1_pref`*. If they are incomparable according to* `c1_pref`*, then compare them using* `c2_pref`*"*. Consider the following program:

```
prioritized(C1,C2)(T1,T2) :- C1(T1,T2).
prioritized(C1,C2)(T1,T2) :- indifferent(C1)(T1,T2),
                             C2(T1,T2).
indifferent(C)(T1,T2) :- not C(T1,T2), not C(T2,T1).
```

The above predicates can be partially applied. As a result, the following query:

```
?- winnow(prioritized(c1_pref,c2_pref),r,T).
```

will return the most preferred tuples from the relation `r` using the prioritized composition of the relations `c1_pref` and `c2_pref`.

The higher-order approach [4, 5], goes beyond most disadvantages of the framework of Chomicki. It allows the definition of *extrinsic* preferences, that is preference relations that depend not only on the values that appear in the tuples, but also on external relations (e.g., *"a tuple t_1 is preferred from t_2 if t_1 belongs to relation p while t_2 belongs to relation q"*). It allows the definition of additional preference compositions (such as the so-called *Pareto* and *Lexicographic* compositions), additional preference operators (such as an operator with the intuitive meaning *"find the second most preferred objects according to this preference"*). We can also define preferences over recursively defined realations and preferences over relations that use preference operators. In general, the use of higher-order logic programming provides a uniform framework in which relations, preferences between tuples, preferences between sets of tuples and operations on preferences are expressed in the same, higher-order logic programming language. Finally, this qualitative approach can be used for implementing practical query systems outside the realm of logic programming and relational databases [23].

3.3 Optimizing Preferential Higher-Order Logic Programs

The higher-order programs of our interest can be evaluated by standard higher-order programming systems. We undertake an implementation[1] of the higer-order framework in the logic programming language HiLog [6] using the XSB System [21]. A naive, unoptimized implementation is relatively straightforward.

The performance of our implementation can be enhanced with several generic and specialized optimization techniques. Among these techniques, we propose [22] and implement[2] *Predicate Specialization*, a novel program transformation technique for translating higher-order programs into first-order ones. The technique

[1] cf. `http://bitbucket.org/antru/holppref`
[2] cf. `http://bitbucket.org/antru/firstify`

is based on *Partial Evaluation* [12]; it gets as input a higher-order logic program P and a goal G and returns a first-order logic program P' and a goal G', such that the computations of $\mathsf{P} \cup \{\mathsf{G}\}$ and $\mathsf{P}' \cup \{\mathsf{G}'\}$ return the same answer set. It works for a well-defined fragment of higher-order logic programs (that is *definitional $\mathcal{H}$ programs*) and for input goals that do not contain predicate variables. In a nutshell, a definitional $\mathcal{H}$ program uses a controlled form of partial applications (to ensure termination) and does not allow existential predicate variables in the bodies of the clauses (to ensure a first-order result). Finally, contrary to other first-order reduction techiques such as *defunctionalization* [16, 24], the resulting programs of Predicate Specialization does not have additional data structures, and as a result the overall program execution time can be reduced.

Example 6. Consider the program P and the query G of Example 3. By applying Predicate Specialization to P according to G we get the following program:

```
winnow1(T) :- movie(T), not bypassed2(T).
bypassed2(T) :- movie(Z), c_pref(Z,T).
```

The initial query G can now be stated in the transformed program as follows:

```
?-winnow1(T).
```

Notice that both queries return the same set of answers, and that this program is *specialized* according to the relation `movie` and the preference relation `c_pref`. Notice that the resulting program is a first-order program due to the fact that the initial query does not contain any predicate variables.

Apart from Predicate Specialization, we used other optimization techiques as well. For optimizing the first-order programs obtained by Predicate Specialization we used *tabling*, a well-known logic programming optimization that avoids re-evaluation of tabled predicates by storing their already computed answers. In addition, for optimizing higher-order logic programs that express preferences over sets we implemented two optimization techiques from Zhang and Chomicki [25], namely *superpreference* and *M-relation*. Each technique uses a mechanism for pruning the set of the candidate k-subsets. Intuitively, superpreference removes tuples that will not contribute to the production of any best k-subset, while the M-relation "groups together" tuples that are exchangeable with respect to a given set preference. Finally, we conduct a series of experiments that highlight the feasibility of the higher-order logic programming framework and the effectiveness of these optimizations.

4 Conclusions and Future Work

This dissertation contributes to the area of preference representation and our results can be perceived as logical frameworks for expressing and manipulating preferences. More specifically, we propose two approaches for expressing preference, namely infinite-valued and higher-order logic programming. Our approaches attempt to overcome some shortcomings of existing approaches in the domain of representation of quantitative and qualitative preferences.

Regading our infinite-valued approach, possibly the most interesting future direction is the addition of negation-as-failure to PrefLog. It has been demonstrated [19] that the meaning of negation can also be captured using the infinite-valued domain $\mathbb{V}$ that we adopted for defining the semantics of PrefLog. It would be interesting to investigate how the preference operators of PrefLog could coexist with negation in a unified framework. Regading our higher-order approach, we believe that it would be very interesting to study the properties and possible evaluation techniques of the higher-order language of our framework, other than Predicate Specialization. For example, it would be interesting to investigate bottom-up proof procedures or other optimizations.

References

[1] Agarwal, R.: A Framework for Expressing Prioritized Constraints Using Infinitesimal Logic. Master's thesis, University of Victoria, Canada (2005)

[2] Agarwal, R., Wadge, W.W.: The lazy evaluation of infinitesimal logic expressions. In: Proceedings of The 2005 International Conference on Programming Languages and Compilers, PLC 2005, Las Vegas, Nevada, USA, June 27-30, 2005. pp. 3–7. CSREA Press (2005)

[3] Agrawal, R., Wimmers, E.L.: A framework for expressing and combining preferences. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA. pp. 297–306. ACM (2000)

[4] Charalambidis, A., Rondogiannis, P., Troumpoukis, A.: Higher-order logic programming: an expressive language for representing qualitative preferences. In: Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016. pp. 24–37. ACM (2016)

[5] Charalambidis, A., Rondogiannis, P., Troumpoukis, A.: Higher-order logic programming: An expressive language for representing qualitative preferences. Sci. Comput. Program. 155, 173–197 (2018)

[6] Chen, W., Kifer, M., Warren, D.S.: HILOG: A foundation for higher-order logic programming. J. Log. Program. 15(3), 187–230 (1993)

[7] Chomicki, J.: Preference formulas in relational queries. ACM Trans. Database Syst. 28(4), 427–466 (2003)

[8] Cui, B., Swift, T.: Preference logic grammars: Fixed point semantics and application to data standardization. Artif. Intell. 138(1-2), 117–147 (2002)

[9] Delgrande, J.P., Schaub, T., Tompits, H., Wang, K.: A classification and survey of preference handling approaches in nonmonotonic reasoning. Computational Intelligence 20(2), 308–334 (2004)

[10] Domshlak, C., Hüllermeier, E., Kaci, S., Prade, H.: Preferences in AI: an overview. Artif. Intell. 175(7-8), 1037–1052 (2011)

[11] Ésik, Z., Rondogiannis, P.: A fixed point theorem for non-monotonic functions. Theor. Comput. Sci. 574, 18–38 (2015)

[12] Gallagher, J.P.: Tutorial on specialisation of logic programs. In: Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'93, Copenhagen, Denmark, June 14-16, 1993. pp. 88–98. ACM (1993)
[13] Govindarajan, K., Jayaraman, B., Mantha, S.: Preference logic programming. In: Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995. pp. 731–745. MIT Press (1995)
[14] Guo, H., Jayaraman, B.: Logic programming with solution preferences. J. Log. Algebr. Program. 78(1), 1–21 (2008)
[15] Koutrika, G., Ioannidis, Y.E.: Personalization of queries in database systems. In: Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA. pp. 597–608. IEEE Computer Society (2004)
[16] Reynolds, J.C.: Definitional interpreters for higher-order programming languages. Higher-Order and Symbolic Computation 11(4), 363–397 (1998)
[17] Rondogiannis, P., Troumpoukis, A.: The infinite-valued semantics: overview, recent results and future directions. Journal of Applied Non-Classical Logics 23(1-2), 213–228 (2013)
[18] Rondogiannis, P., Troumpoukis, A.: Expressing preferences in logic programming using an infinite-valued logic. In: Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, July 14-16, 2015. pp. 208–219. ACM (2015)
[19] Rondogiannis, P., Wadge, W.W.: Minimum model semantics for logic programs with negation-as-failure. ACM Trans. Comput. Log. 6(2), 441–467 (2005)
[20] Stefanidis, K., Koutrika, G., Pitoura, E.: A survey on representation, composition and application of preferences in database systems. ACM Trans. Database Syst. 36(3), 19:1–19:45 (2011)
[21] Swift, T., Warren, D.S.: XSB: extending prolog with tabled logic programming. TPLP 12(1-2), 157–187 (2012)
[22] Troumpoukis, A., Charalambidis, A.: Predicate specialization for definitional higher-order logic programs. In: Logic-Based Program Synthesis and Transformation - 28th International Symposium, LOPSTR 2018, Frankfurt/Main, Germany, September 4-6, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11408, pp. 132–147. Springer (2018)
[23] Troumpoukis, A., Konstantopoulos, S., Charalambidis, A.: An extension of SPARQL for expressing qualitative preferences. In: The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10587, pp. 711–727. Springer (2017)
[24] Warren, D.H.D.: Higher-order extensions to Prolog: Are they needed? In: Machine Intelligence, vol. 10, pp. 441–454. Ellis Horwood (1982)
[25] Zhang, X., Chomicki, J.: Preference queries over sets. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany. pp. 1019–1030. IEEE Computer Society (2011)