Querying with Intrinsic Preferences*

Jan Chomicki

Dept. of Computer Science and Engineering, University at Buffalo, Buffalo, NY 14260-2000, chomicki@cse.buffalo.edu

Abstract. The handling of user preferences is becoming an increasingly important issue in present-day information systems. Among others, preferences are used for *information filtering and extraction* to reduce the volume of data presented to the user. They are also used to keep track of *user profiles* and formulate *policies* to improve and automate decision making. We propose a logical framework for formulating preferences and its embedding into relational query languages. The framework is simple, and entirely neutral with respect to the properties of preferences. It makes it possible to formulate different kinds of preferences and to use preferences in querying databases. We demonstrate the usefulness of the framework through numerous examples.

1 Introduction

The handling of user preferences is becoming an increasingly important issue in present-day information systems. Among others, preferences are used for *information filtering and extraction* to reduce the volume of data presented to the user. They are also used to keep track of *user profiles* and formulate *policies* to improve and automate decision making.

The research literature on preferences is extensive. It encompasses preference logics [23,19,13], preference reasoning [24,22,4], prioritized nonmonotonic reasoning and logic programming [5,6,21] and decision theory [7,8] (the list is by no means exhaustive). However, only a few papers [18,3,11,2,14,17] address the issue of user preferences in the context of database queries. Two different approaches are pursued: qualitative and quantitative. In the *qualitative* approach [18,3,11,17], the preferences between tuples in the answer to a query are specified directly, typically using binary *preference relations.*

Example 1. We introduce here one of the examples used throughout the paper. Consider the relation $Book(ISBN, Vendor, Price)$ and the following preference relation $\succ_1$ between $Book$ tuples:

> *if two tuples have the same ISBN and different Price, prefer the one with the lower Price.*

* To appear in EDBT 2002.

Consider the following instance r_1 of *Book*

ISBN	*Vendor*	*Price*
0679726691	BooksForLess	$14.75
0679726691	LowestPrices	$13.50
0679726691	QualityBooks	$18.80
0062059041	BooksForLess	$7.30
0374164770	LowestPrices	$21.88

Then clearly the second tuple is preferred to the first one which in turn is preferred to the third one. There is no preference defined between any of those three tuples and the remaining tuples.

In the *quantitative* approach [2,14], preferences are specified indirectly using *scoring functions* that associate a numeric score with every tuple of the query answer. Then a tuple $\bar{t}_1$ is preferred to a tuple $\bar{t}_2$ iff the score of $\bar{t}_1$ is higher than the score of $\bar{t}_2$. The qualitative approach is strictly more general than the quantitative one, since one can define preference relations in terms of scoring functions (if the latter are explicitly given), while not every intuitively plausible preference relation can be captured by scoring functions.

Example 2. There is no scoring function that captures the preference relation described in Example 1. Since there is no preference defined between any of the first three tuples and the fourth one, the score of the fourth tuple should be equal to all of the scores of the first three tuples. But this implies that the scores of the first three tuples are the same, which is not possible since the second tuple is preferred to the first one which in turn is preferred to the third one.

This lack of expressiveness of the quantitative approach is well known in utility theory [7,8].

In the present paper, we contribute to the qualitative approach by defining a logical framework for formulating preferences and its embedding into relational query languages.

We believe that combining preferences with queries is very natural and useful. The applications in which user preferences are prominent will benefit from applying the modern database technology. For example, in decision-making applications databases may be used to store the space of possible configurations. Also, the use of a full-fledged query language makes it possible to formulate complex decision problems, a feature missing from most previous, non-database, approaches to preferences. For example, the formulation of the problem may now involve quantifiers, grouping, or aggregation. At the same time by explicitly addressing the technical issues involved in querying with preferences present-day DBMS may expand their scope.

The framework presented in this paper consists of two parts: a formal first-order logic notation for specifying preferences and an embedding of preferences into relational query languages. In this way both abstract properties of preferences (like asymmetry or transitivity) and evaluation of preference queries can be studied to a large degree separately.

Preferences are defined using binary *preference relations* between tuples. Preference relations are specified using first-order formulas. We focus mostly on *intrinsic* preference formulas. Such formulas can refer only to built-in predicates. In that way we capture preferences that are based only on the values occuring in tuples, not on other properties like membership of tuples in database relations. We show how the latter kind of preferences, called *extrinsic*, can also be simulated in our framework in some cases.

We propose a new relational algebra operator called *winnow* that selects from its argument relation the *most preferred tuples* according to the given preference relation. Although the winnow operator can be expressed using other operators of relational algebra, by considering it on its own we can on one hand focus on the abstract properties of preference relations (e.g., transitivity) and on the other, study special evaluation and optimization techniques for the winnow operator itself. The winnow operator can also be expressed in SQL.

We want to capture many different varieties of preference and related notions: *unconditional* vs. *conditional* preferences, *nested* and *hierarchical* preferences, *groupwise* preferences, *indifference*, *iterated* preferences and *ranking*, and *integrity constraints* and *vetoes*.

In Section 2, we define the basic concepts of preference relation, preference formula, and the winnow operator. In Section 3, we study the basic properties of the above concepts. In Section 4, we explore the composition of preferences. In Section 5, we show how the winnow operator together with other operators of relational algebra makes it possible to express integrity constraints and extrinsic preferences. In Section 6, we show how iterating the winnow operator provides a ranking of tuples and introduce a weak version of the winnow operator that is helpful for non-asymmetric preference relations. We discuss related work in Section 7 and conclude with a brief discussion of further work in Section 8.

2 Basic notions

We are working in the context of the relational model of data. We assume two infinite domains: D (uninterpreted constants) and N (numbers). We do not distinguish between different numeric domains, since it is not necessary for the present paper. We assume that database instances are finite (this is important). Additionally, we have the standard built-in predicates. In the paper, we will move freely between relational algebra and SQL.

2.1 Basic definitions

Preference formulas are used to define binary preference relations.

Definition 1. *Given $U = (U_1, \ldots, U_k)$ such that U_i, $1 \leq i \leq k$, is either D or N, a relation $\succ$ is a* preference relation over U *if it is a subset of $(U_1 \times \cdots \times U_k) \times (U_1 \times \cdots \times U_k)$.*

Intuitively, $\succ$ will be a binary relation between pairs of tuples from the same (database) relation. We say that a tuple $\bar{t}_1$ *dominates* a tuple $\bar{t}_2$ in $\succ$ if $\bar{t}_1 \succ \bar{t}_2$.

Typical properties of the relation $\succ$ include:

- *irreflexivity*: $\forall x. x \not\succ x$,
- *asymmetry*: $\forall x, y. x \succ y \Rightarrow y \not\succ x$,
- *transitivity*: $\forall x, y, z. (x \succ y \wedge y \succ z) \Rightarrow x \succ z$.

The relation $\succ$ is a *strict partial order* if it is irreflexive, asymmetric and transitive. At this point, we do not assume any properties of $\succ$, although in most applications it will be at least a strict partial order.

Definition 2. *A* preference formula (pf) $C(\bar{t}_1, \bar{t}_2)$ *is a first-order formula defining a preference relation* $\succ$ *in the standard sense, namely*

$$\bar{t}_1 \succ \bar{t}_2 \text{ iff } C(\bar{t}_1, \bar{t}_2).$$

An intrinsic preference formula (ipf) *is a preference formula that uses only built-in predicates.*

Ipfs can refer to equality ($=$) and inequality ($\neq$) when comparing values that are uninterpreted constants, and to the standard set of built-in arithmetic comparison operators when referring to numeric values (there no function symbols). We will call an ipf that references only arithmetic comparisons ($=, \neq, <, >, \leq, \geq$) *arithmetical*. Without loss of generality, we will assume that ipfs are in DNF (Disjunctive Normal Form) and quantifier-free (the theories involving the above predicates admit quantifier elimination).

In this paper, we mostly restrict ourselves to ipfs and preference relations defined by such formulas. The main reason is that ipfs define *fixed*, although possibly, infinite relations. As a result, they are computationally easier and more amenable to syntactic manipulation that general pfs. For instance, transitively closing an ipf results in a finite formula (Theorem 4), which is typically not the case for pfs. However, we formulate in full generality the results that hold for arbitrary pfs.

We define now an algebraic operator that picks from a given relation the set of the *most preferred tuples*, according to a given preference formula.

Definition 3. *If R is a relation schema and C a preference formula defining a preference relation $\succ$ over R, then the* winnow operator *is written as $\omega_C(R)$, and for every instance r of R:*

$$\omega_C(r) = \{\bar{t} \in r | \neg \exists \bar{t'} \in r.\ \bar{t'} \succ \bar{t}\}.$$

A preference query is a relational algebra query containing at least one occurrence of the winnow operator.

2.2 Examples

The first example illustrates how preference queries are applied to *information extraction*: here obtaining the best price of a given book.

Example 3. Consider the relation $Book(ISBN, Vendor, Price)$ from Example 1. The preference relation $\succ_1$ from this example can be defined using the formula C_1:

$$(i,v,p) \succ_1 (i',v',p') \equiv i = i' \wedge p < p'.$$

The answer to the preference query $\omega_{C_1}(Book)$ provides for every book the information about the vendors offering the lowest price for that book. For the given instance r_1 of $Book$, applying the winnow operator ω_{C_1} returns the tuples

ISBN	*Vendor*	*Price*
0679726691	LowestPrices	\$13.50
0062059041	BooksForLess	\$7.30
0374164770	LowestPrices	\$21.88

Note that in the above example, the preferences are applied *groupwise*: separately for each book. Note also that due to the properties of $<$, the preference relation $\succ_1$ is irreflexive, asymmetric and transitive.

The second example illustrates how preference queries are used in *automated decision making* to obtain the most desirable solution to a (very simple) configuration problem.

Example 4. Consider two relations $Wine(Name, Type)$ and $Dish(Name, Type)$ and a view $Meal$ that contains possible meal configurations

```
CREATE VIEW Meal(Dish,DishType,Wine,WineType) AS
  SELECT * FROM Wine, Dish;
```

Now the preference for white wine in the presence of fish and for red wine in the presence of meat can be expressed as the following preference formula C_2 over $Meal$:

$$\begin{aligned}(d,dt,w,wt) \succ_2 (d',dt',w',wt') \equiv\ & (d = d' \wedge dt = '\text{fish}' \wedge wt = '\text{white}' \\ & \wedge dt' = '\text{fish}' \wedge wt' = '\text{red}') \\ & \vee (d = d' \wedge dt = '\text{meat}' \wedge wt = '\text{red}' \\ & \wedge dt' = '\text{meat}' \wedge wt' = '\text{white}')\end{aligned}$$

Notice that this will force any white wine to be preferred over any red wine for fish, and just the opposite for meat. For other kinds of dishes, no preference is indicated. Consider now the preference query $\omega_{C_2}(Meal)$. It will pick the most preferred meals, according to the above-stated preferences. Notice that in the absence of any white wine, red wine (or some other kind of wine, e.g., rosé) can be selected for fish.

The above preferences are conditional, since they depend on the type of the dish being considered. Note that the relation $\succ_2$ in this example is irreflexive and asymmetric. Transitivity is obtained trivially because the chains of $\succ_2$ are of length at most 2. Note also that the preference relation is defined without referring to any domain order.

The unconditional preference for red wine for any kind of meal can also be defined as a first-order formula C_3:

$$(d, dt, w, wt) \succ_3 (d', dt', w', wt') \equiv d = d' \wedge wt = '\mathrm{red}' \wedge wt' \neq '\mathrm{red}'.$$

3 Properties of preference queries

3.1 Preference relations

Since pfs can be essentially arbitrary formulas, no properties of preference relations can be assumed. So our framework is entirely neutral in this respect.

In the examples above, the preference relations were strict partial orders. This is likely to be the case for most applications of preference queries. However, there are cases where such relations fail to satisfy one of the properties of partial orders. We will see in Section 5 when irreflexivity fails. For asymmetry: We may have two tuples $\bar{t}_1$ and $\bar{t}_2$ such that $\bar{t}_1 \succ \bar{t}_2$ and $\bar{t}_2 \succ \bar{t}_1$ simply because we may have one reason to prefer $\bar{t}_1$ over $\bar{t}_2$ and another reason to prefer $\bar{t}_2$ over $\bar{t}_1$. Similarly, transitivity is not always guaranteed [15,19,7,13]. For example, $\bar{t}_1$ may be preferred over $\bar{t}_2$ and $\bar{t}_2$ over $\bar{t}_3$, but the gap between $\bar{t}_1$ and $\bar{t}_3$ with respect to some heretofore ignored property may be so large as to prevent preferring $\bar{t}_1$ over $\bar{t}_3$. Or, transitivity may have to be abandoned to prevent cycles in preferences.

However, it is not difficult to check the properties of a preference relation defined using an ipf.

Theorem 1. *If a preference relation is defined using an arithmetical ipf, it can be checked for irreflexivity, asymmetry and transitivity in PTIME.*

Proof: We discuss asymmetry, the remaining properties can be handled in a similar way. If $\bar{t}_1 \succ \bar{t}_2$ is defined as $D_1 \vee \ldots \vee D_m$ and $\bar{t}_2 \succ \bar{t}_1$ as $D'_1 \vee \ldots \vee D'_m$, we can write down the negation of asymmetry as $(D_1 \vee \ldots \vee D_m) \wedge (D'_1 \vee \ldots \vee D'_m)$. The satisfiability of this formula can be checked in PTIME using the methods of [12]. □

Theorem 2. *If a preference relation over R defined using a pf C is a strict partial order, then for every finite, nonempty instance r of R, $\omega_C(r)$ is nonempty.*

If the properties of strict partial orders are not satisfied, then Theorem 2 may fail to hold and the winnow operator may return an empty set, even though the relation to which it is applied is nonempty. For instance, if $r_0 = \{\bar{t}_0\}$ and $\bar{t}_0 \succ \bar{t}_0$ (violation of irreflexivity), then the winnow operator applied to r_0 returns an empty set. Similarly, if two tuples are involved in a violation of asymmetry, they may block each other from appearing in the result of the winnow operator. Also, if the relation r is infinite, it may happen that $\omega_C(r) = \emptyset$, for example if r

contains all natural numbers and the preference relation is the standard ordering $>$.

We conclude this subsection by noting that there is a natural notion of *indifference* associated with our approach: two tuples $\bar{t}_1$ and $\bar{t}_2$ are *indifferent* ($\bar{t}_1 \sim \bar{t}_2$) if neither is preferred to the other one, i.e., $\bar{t}_1 \not\succ \bar{t}_2$ and $\bar{t}_2 \not\succ \bar{t}_1$.

Proposition 1. *For every pf C, every relation r and every tuple $\bar{t}_1, \bar{t}_2 \in \omega_C(r)$, we have $\bar{t}_1 = \bar{t}_2$ or $\bar{t}_1 \sim \bar{t}_2$.*

It is a well-known result in decision theory [7,8] that in order for a preference relation to be representable using scoring functions the corresponding indifference relation (defined as above) has to be *transitive*. This is not the case for the preference relation $\succ_1$ defined in Example 1.

3.2 The winnow operator

The winnow operator $\omega_C(R)$ such that $C = D_1 \vee \ldots \vee D_k$ is an ipf can be expressed in relational algebra, and thus does not add any expressive power to it. To see that notice that each D_i, $i = 1, \ldots, k$, is a formula over free variables $\bar{t}_1$ and $\bar{t}_2$. It can be viewed as a conjunction $D_i \equiv \phi_i \wedge \psi_i \wedge \gamma_i$ where ϕ_i refers only to the variables of $\bar{t}_1$, ψ_i to the variables of $\bar{t}_2$, and γ_i to the variables of both $\bar{t}_1$ and $\bar{t}_2$. The formula ϕ_i has an obvious translation to a selection condition Φ_i over R, and the formula ψ_i a similar translation to a selection condition Ψ_i over $\varrho(R)$, where ϱ is a renaming of R. The formula γ_i can similarly be translated to a join condition Γ_i over R and $\varrho(R)$. Then

$$\omega_C(R) = \varrho^{-1}(\varrho(R) - \pi_{\varrho(R)}(\bigcup_{i=1}^{k}(\sigma_{\Phi_i}(R) \underset{\Gamma_i}{\bowtie} \sigma_{\Psi_i}(\varrho(R)))))$$

where ϱ^{-1} is the inverse of the renaming ϱ.

However, the use of the winnow operator makes possible a clean separation of preference formulas from other aspects of the query. This has several advantages. First, the properties of preference relations can be studied in an abstract way, as demonstrated in this section and the next. Second, specialized query evaluation methods for the winnow operator can be developed. Third, algebraic properties of that operator can be formulated, in order to be used in query optimization.

To see the importance of the second point, note that a simple nested loops strategy is sufficient for evaluating the winnow operator. This is not at all obvious from considering the equivalent relational algebra expression. For restricted cases of the winnow operator, e.g., skylines [3], even more efficient evaluation strategies may be available. For the third point, we identify in Theorem 3 below a condition under which the winnow operator and a relational algebra selection commute. This is helpful for pushing selections past winnow operators in preference queries.

Theorem 3. *If the formula $\forall(C_1(\bar{t}_2) \wedge C_2(\bar{t}_1, \bar{t}_2)) \Rightarrow C_1(\bar{t}_1)$ (where C_1 is a selection condition and C_2 is a preference formula) is valid, then*

$$\sigma_{C_1}(\omega_{C_2}(R)) = \omega_{C_2}(\sigma_{C_1}(R)).$$

If the preference formula C_2 in the above theorem is an arithmetical ipf and the selection condition C_1 refers only to the arithmetic comparison predicates, then checking the validity of the formula

$$\forall (C_1(\bar{t}_2) \wedge C_2(\bar{t}_1, \bar{t}_2)) \Rightarrow C_1(\bar{t}_1)$$

can be done in PTIME.

Finally, we note that the winnow operator $\omega_C(R)$ such that C is an arbitrary preference formula (not necessarily an ipf) is still first-order definable. However, since the preference formula can now reference database relations, the relational algebra formulation may be considerably more complicated.

4 Composition of preferences

In this section, we study several ways of composing preference relations. Since in our approach such relations are defined using preference formulas, composing them will amount to generating a formula defining the result of the composition. We will consider Boolean composition, transitive closure and prioritized composition.

4.1 Boolean composition

Union, intersection and difference of preference relations are obviously captured by the Boolean operations on the corresponding preference formulas. The following table summarizes the preservation of properties of relations by the appropriate boolean composition operator.

	Union	Intersection	Difference
Irreflexivity	Yes	Yes	Yes
Asymmetry	No	Yes	Yes
Transitivity	No	Yes	No

4.2 Transitive closure

We have seen an example (Example 1) of a preference relation that is already transitive. However, there are cases when we expect the preference relation to be the *transitive closure* of another preference relation which is not transitive.

Example 5. Consider the following relation:

$$x \succ y \equiv x = a \wedge y = b \vee x = b \wedge y = c.$$

In this relation, a and c are not related though there are contexts in which this might be natural. (Assume I prefer to walk than to drive, and to drive than to ride a bus. Thus, I also prefer to walk than to ride a bus.)

In our framework, we can specify the preference relation $\succ^*$ to be the *transitive closure* of another preference relation $\succ$ defined using a first-order formula. This is similar to transitive closure queries in relational databases. However, there is an important difference. In databases, we are computing the transitive closure of a *finite* relation, while here we are transitively closing an infinite relation defined using a first-order formula.

Formally, assuming that the underlying preference relation is $\succ$, the preference relation $\succ^*$ is now defined as

$$\bar{t}_1 \succ^* \bar{t}_2 \text{ iff } \bar{t}_1 \succ^n \bar{t}_2 \text{ for some } n \geq 0,$$

where:

$$\begin{array}{l} \bar{t}_1 \succ^1 \bar{t}_2 \equiv \bar{t}_1 \succ \bar{t}_2 \\ \bar{t}_1 \succ^{n+1} \bar{t}_2 \equiv \exists \bar{t}_3.\ \bar{t}_1 \succ \bar{t}_3 \wedge \bar{t}_3 \succ^n \bar{t}_2. \end{array}$$

Clearly, in general such an approach leads to infinite formulas. However, in many important cases this does not happen.

Theorem 4. *If a preference relation $\succ$ is defined using an arithmetical ipf, the transitive closure $\succ^*$ of $\succ$ is also defined using an arithmetical ipf and that definition can be effectively obtained.*

Proof: The computation of the transitive closure can in this case be formulated as the evaluation of Datalog with order or gap-order (for integers) constraints. Such an evaluation terminates [16,20] and its result represents the desired formula. $\square$

An analogous result holds if instead of arithmetic comparisons we consider equality constraints over an infinite domain [16].

Example 6. Continuing Example 5, we obtain the following preference relation $\succ^*$ by transitively closing $\succ$:

$$x \succ^* y \equiv x = a \wedge y = b \vee x = b \wedge y = c \vee x = a \wedge y = c.$$

Theorem 4 is not in conflict with the well-known non-first order definability of transitive closure on finite structures. In the latter case it is shown that there is no finite first-order formula expressing transitive closure for arbitrary (finite) binary relations. In Theorem 4 the relation to be closed, although possibly infinite, is fixed (since it is defined using the given ipf). In particular, given an encoding of a fixed finite binary relation using an ipf, the transitive closure of this relation is defined using another ipf.

The transitive closure of a irreflexive (resp. asymmetric) preference relation may fail to be irreflexive (resp. asymmetric).

4.3 Preference hierarchies

It is often the case that preferences form hierarchies. For instance, I may have a general preference for red wine but in specific cases, e.g., when eating fish, this preference is overridden by the one for white wine. Also a preference for less expensive books (Example 1) can be overridden by a preference for certain vendors.

Definition 4. *Consider two preference relations* $\succ_1$ *and* $\succ_2$ *defined over the same schema* U*. The* prioritized composition $\succ_{1,2}=\succ_1 \rhd \succ_2$ *of* $\succ_1$ *and* $\succ_2$ *is defined as:*

$$\bar{t}_1 \succ_{1,2} \bar{t}_2 \equiv \bar{t}_1 \succ_1 \bar{t}_2 \vee (\bar{t}_2 \not\succ_1 \bar{t}_1 \wedge \bar{t}_1 \succ_2 \bar{t}_2).$$

Example 7. Continuing Example 1, instead of the preference relation $\succ_1$ defined there we consider the relation $\succ_0 \rhd \succ_1$ where $\succ_0$ is defined by the following formula C_0:

$$(i,v,p) \succ_0 (i',v',p') \equiv i = i' \wedge v = '\text{BooksForLess}' \wedge v' = '\text{LowestPrices}'.$$

Assume the relation $\succ_{0,1}=\succ_0 \rhd \succ_1$ is defined by a formula $C_{0,1}$ (this formula is easily obtained from the formulas C_0 and C_1 by substitution). Then $\omega_{C_{0,1}}(r_1)$ returns the following tuples

ISBN	*Vendor*	*Price*
0679726691	BooksForLess	\$14.75
0062059041	BooksForLess	\$7.30
0374164770	LowestPrices	\$21.88

Note that now a more expensive copy of the first book is preferred, due to the preference for 'BooksForLess' over 'LowestPrices'. However, 'BooksForLess' does not offer the last book, and that's why the copy offered by 'LowestPrices' is preferred.

Theorem 5. *If* $\succ_1$ *and* $\succ_2$ *are defined using intrinsic preference formulas, so is* $\succ_{1,2}$*. If* $\succ_1$ *and* $\succ_2$ *are both irreflexive or asymmetric, so is* $\succ_{1,2}$*.*

However, a relation defined as the prioritized composition of two transitive preference relations does not have to be transitive.

Example 8. Consider the following preference relations:

$$a \succ_1 b, b \succ_2 c.$$

Both $\succ_1$ and $\succ_2$ are trivially transitive. However, $\succ_1 \rhd \succ_2$ is not.

Theorem 6. *Prioritized composition is associative:*

$$(\succ_1 \rhd \succ_2) \rhd \succ_3 \equiv \succ_1 \rhd (\succ_2 \rhd \succ_3).$$

Thanks to the associativity of $\rhd$, the above construction can be generalized to an arbitrary finite partial order between preference relations. Such an order can be viewed as a graph in which the nodes consist of preference relations and the edges represent relative priorities (there would be an edge $(\succ_1, \succ_2)$ in the situation described above). To encode this graph as a single preference relation, one would construct first the definitions corresponding to individual paths from roots to leaves, and then take a disjunction of all such definitions.

There may be other ways of combining preferences. For instance, preference relations defined on individual database relations may induce other preferences defined on the Cartesian product of the database relations. In general, any first-order definable way of composing preference relations leads to first-order preference formulas, which in turn can be used as parameters of the winnow operator. The composition does not even have to be first-order definable, as long as it produces a (first-order) preference formula.

5 More expressive preferences

We show here that the winnow operator when used together with other operators of the relational algebra can express more complex decision problems involving preferences. We consider the following: integrity constraints, extrinsic preferences, and aggregation.

5.1 Integrity constraints

There are cases when we wish to impose a constraint on the result of the winnow operator. In Example 1, we may say that we are interested only in the books under \$20. In Example 4, we may restrict our attention only to the meat or fish dishes (note that currently the dishes that are not meat or fish do not have a preferred kind of wine). In the same example, we may ask for a specific number of meal recommendations.

In general, we need to distinguish between *local* and *global* constraints. A local constraint imposes a condition on the components of a single tuple, for instance `Book.Price<$20`. A global constraint imposes a condition on a set of tuples. The first two examples above are local constraints; the third is global. To satisfy a global constraint on the result of the winnow operator, one would have to construct a maximal subset of this answer that satisfies the constraint. Since in general there may be more than one such subset, the required construction cannot be described using a single relational algebra query. On the other hand, local constraints are easily handled, since they can be expressed using selection.

Example 9. Consider Example 1. The preference formula C_1 captures the preference for getting the same book cheaper. If we want to limit ourselves to books that cost less than \$20, we can use the following relational algebra query:

$$\sigma_{Price<\$20}(\omega_{C_1}(Book)).$$

According to Theorem 3, this query is equivalent to the query

$$\omega_{C_1}(\sigma_{Price<\$20}(Book))$$

which may be easier to evaluate, since the selection is applied directly to a database table. On the other hand, if we ask for books that cost at least \$20, the corresponding selection will not commute with the winnow operator. Intuitively, such a selection can eliminate some of the best deals on books. So in general it matters whether the integrity constraints are imposed before or after applying the winnow operator.

A *veto* expresses a prohibition on the presence of a specific set of values in the elements of the answer to a preference query and thus can be viewed as a local constraint. To veto a specific tuple $w = (a_1, \ldots, a_n)$ in a relation S (which can be defined by a preference query) of arity n, we write the selection:

$$\sigma_{A_1 \neq a_1 \vee \cdots \vee A_n \neq a_n}(S).$$

5.2 Intrinsic vs. extrinsic preferences

So far we have talked only about *intrinsic* preference formulas. Such formulas establish the preference relation between two tuples purely on the basis of the values occurring in those tuples. *Extrinsic* preference formulas may refer not only to built-in predicates but also to other constructs, e.g., database relations. In general, extrinsic preferences can use a variety of criteria: properties of the relations from which the tuples were selected, properties of other relations, or comparisons of aggregate values, and do not even have to be defined using first-order formulas.

It is possible to express some extrinsic preferences using the winnow operator together with other relational algebra operators using the following multi-step strategy:

1. using a relational query, combine all the information relevant for the preference in a single relation,
2. apply the appropriate winnow operator to this relation,
3. project out the extra columns introduced in the first step.

The following example demonstrates the above strategy, as well as the use of aggregation for the formulation of preferences.

Example 10. Consider again the relation $Book(ISBN, Vendor, Price)$. Suppose for each part a preferred vendor (there may be more than one) is a vendor that sells the *maximum total* number of books. Clearly, this is an extrinsic preference since it cannot be established solely by comparing pairs of tuples from this relation. However, we can provide the required aggregate values and connect them with individual parts through new, separate views:

```
CREATE VIEW BookNum(Vendor,Num) AS
  SELECT B1.Vendor, COUNT(DISTINCT B1.ISBN)
  FROM Book B1
  GROUP BY B1.Vendor;

CREATE VIEW ExtBook(ISBN,Vendor,Num) AS
  SELECT B1.ISBN, B1.Vendor, BN.Num
  FROM Book B1, BookNum BN
  WHERE B1.Vendor=BN.Vendor;
```

Now the extrinsic preference is captured by the query

$$\pi_{ISBN,Vendor}(\omega_{C_5}(ExtBook))$$

where the preference formula C_5 is defined as follows:

$$(i,v,n) \succ_5 (i',v',n') \equiv i = i' \wedge n > n'.$$

Example 11. To see another example of extrinsic preference, consider the situation in which we prefer any tuple from a relation R over any tuple from a relation S. Notice that this is truly an extrinsic preference, since it is based on where the tuples come from and not on their values. It can be handled in our approach by *tagging* the tuples with the appropriate relation names (easily done in relational algebra or SQL) and then defining the preference relation using the tags. If there is a tuple which belongs both to R and S, then the above preference relation will fail to be irreflexive. Note also that an approach similar to tagging was used in Example 4.

6 Iterated preferences and ranking

A natural notion of *ranking* is implicit in our approach. A ranking is defined using *iterated preference.*

Definition 5. *Given a preference relation $\succ$ defined by a pf C, the* n-th iteration *of the winnow operator ω_C in r is defined as:*

$$\begin{array}{l} \omega_C^1(r) = \omega_C(r) \\ \omega_C^{n+1}(r) = \omega_C(r - \bigcup_{1 \leq i \leq n} \omega_C^i(r)) \end{array}$$

For example, the query $\omega_C^2(r)$ computes the set of "second-best" tuples.

Example 12. Continuing Example 1, the query $\omega_{C_1}^2(r_1)$ returns

ISBN	*Vendor*	*Price*
0679726691	BooksForLess	\$14.75

and the query $\omega_{C_1}^3(r_1)$ returns

ISBN	*Vendor*	*Price*
0679726691	QualityBooks	\$18.80

Therefore, by iterating the winnow operator one can *rank* the tuples in the given relation instance.

Theorem 7. *If a preference relation $\succ$ defined by a first-order formula C over R is a strict partial order, then for every finite instance r of R and every tuple $\bar{t} \in r$, there exists an i, $i \geq 1$, such that $\bar{t} \in \omega_C^i(r)$.*

If a preference relation is not a strict partial order, then Theorem 7 may fail to hold. A number of tuples can block one another from appearing in the result of any iteration of the winnow operator. However, even in this case there may be a weaker form of ranking available.

Example 13. Consider Examples 1 and 7. If the preference formula C' is defined as $C_0 \vee C_1$, then the first two tuples of the instance r_1 block each other from appearing in the result of $\omega_{C'}(r_1)$, since according to C_0 the first tuple is preferred to the second but just the opposite is true according to C_1. Intuitively, both those tuples should be preferred to (and ranked higher) than the third tuple. But since neither the first not the second tuple is a member of $\omega_{C'}(r_1)$, none of the first three tuples can be ranked.

We define now a weaker form of the winnow operator that will return all the tuples that are dominated only by the tuples that they dominate themselves. We relax the asymmetry requirement but preserve transitivity.

Definition 6. *If R is a relation schema and C a pf defining a transitive preference relation $\succ$ over R, then the* weak winnow operator *is written as $\psi_C(R)$ and for every instance r of R*

$$\psi_C(r) = \{\bar{t} \in r | \forall \bar{t'} \in r . \bar{t} \succ \bar{t'} \vee \bar{t'} \not\succ \bar{t}\}.$$

Example 14. Considering Example 13, we see that the query $\psi_{C'}(r_1)$ returns now

ISBN	*Vendor*	*Price*
0679726691	BooksForLess	\$14.75
0679726691	LowestPrices	\$13.50
0062059041	BooksForLess	\$7.30
0374164770	LowestPrices	\$21.88

Below we formulate a few properties of the weak winnow operator.

Theorem 8. *If R is a relation schema and C a preference formula defining a transitive preference relation $\succ$ over R, then for every instance r of R, $\psi_C(r)$ is uniquely defined and $\omega_C(r) \subseteq \psi_C(r)$.*

Theorem 9. *For every finite, nonempty relation instance r of R, and a transitive preference relation $\succ$ over R defined by a preference formula C, $\psi_C(r)$ is nonempty.*

One can define the iteration of the weak winnow operator similarly to that of the winnow operator (Definition 5).

Theorem 10. *If a preference relation $\succ$ over R defined by a first-order formula C is transitive, then for every finite instance r of R and for every tuple $\bar{t} \in r$, there exists an i, $i \geq 1$, such that $\bar{t} \in \psi^i_C(r)$.*

7 Related work

7.1 Preference queries

[18] originated the study of *preference queries.* It proposed an extension of the relational calculus in which preferences for tuples satisfying given logical conditions can be expressed. For instance, one could say: *Among the tuples of R*

satisfying Q, I prefer those satisfying P_1; among the latter I prefer those satisfying P_2. Such a specification was to mean the following: Pick the tuples satisfying $Q \wedge P_1 \wedge P_2$; if the result is empty, pick the tuples satisfying $Q \wedge P_1 \wedge \neg P_2$; if the result is empty, pick the remaining tuples of R satisfying Q. This can be simulated in our framework as the relational algebra expression $\omega_{C^*}(\sigma_Q(R))$ where C^* is an ipf defined in the following way:

1. obtain the formula C defining a preference relation $\succ$

$$\bar{t}_1 \succ \bar{t}_2 \equiv P_1(\bar{t}_1) \wedge P_2(\bar{t}_1) \wedge P_1(\bar{t}_2) \wedge \neg P_2(\bar{t}_2) \vee P_1(\bar{t}_1) \wedge \neg P_2(\bar{t}_1) \wedge \neg P_1(\bar{t}_2),$$

2. transform C into DNF to obtain an ipf C', and
3. close the result transitively to obtain an ipf C^* defining a transitive preference relation $\succ^*$ (as described in Section 4).

Other kinds of logical conditions from [18] can be similarly expressed in our framework. Maximum/minimum value preferences (as in Example 1) are handled in [18] through the explicit use of aggregate functions. The use of such functions is implicit in the definition of our winnow operator.

Unfortunately, [18] does not contain a formal definition of the proposed language, so a complete comparison with our approach is not possible. It should be noted, however, that the framework of [18] seems unable to capture very simple conditional preferences like the ones in Examples 4 and 5. Also, it can only handle strict partial orders of bounded depth (except in the case where aggregate functions can be used, as in Example 1). Hierarchical or iterated preferences are not considered.

[11] was one of the sources of inspiration for the present paper. It defines *Preference Datalog*: a combination of Datalog and clausally-defined preference relations. Preference Datalog captures, among others, the class of preference queries discussed in [18]. The declarative semantics of Preference Datalog is based on the notion of *preferential consequence*, introduced earlier by the authors in [10]. This semantics requires preferences to be reflexive and transitive. Also, the operational semantics of Preference Datalog uses specialized versions of the standard logic program evaluation methods: bottom-up [11] or top-down [10]. In the context of database queries, the approach proposed in the present paper achieves similar goals to that of [10] and [11], remaining, however, entirely within the relational data model and classical first-order logic. Finally, [10,11] do not address some of the issues we deal with in the present paper like transitive closure of preferences, prioritized composition or iterated preferences (a similar concept to the last one is presented under the name of "relaxation").

[17] discusses Preference SQL, a query language in which preferences between atomic conditions can be stated. For example, I can say that I prefer the book "ABC" with price under \$20 over the same book with price over \$20. However, saying simply that I prefer a lower price on a book (as we do in Example 1) does not seems possible in Preference SQL. The description of Preference SQL in [17] is so brief that a detailed comparison with our proposal is not possible at this point. [3] introduces the *skyline* operator and describes several evaluation

methods for this operator. The skyline is a special case of our winnow operator. It is restricted to use an arithmetical ipf which is a conjunction of pairwise comparisons of corresponding tuple components. So in particular Example 4 does not fit in that framework.

[2] uses quantitative preferences in queries and focuses on the issues arising in combining such preferences. [14] explores in this context the problems of efficient query processing. Since the preferences in this approach are based on comparing the scores of individual tuples under given scoring functions, they have to be intrinsic. However, the simulation of extrinsic preferences using intrinsic ones (Section 5) is not readily available in this approach because the scoring functions are not integrated with the query language. So, for instance, Example 10 cannot be handled. In fact, even for preference relations that satisfy the property of transitivity of the corresponding indifference relation, it is not clear whether the scoring function capturing the preference relation can be defined intrinsically (i.e., the function value be determined solely by the the values of the tuple components). The general construction of a scoring function on the basis of a preference relation [7,8] does not provide such a definition. So the exact expressive power of the quantitative approach to preference queries remains unclear.

7.2 Preferences in logic and artificial intelligence

The papers on *preference logics* [23,19,13] address the issue of capturing the common-sense meaning of preference through appropriate axiomatizations. Preferences are defined on formulas, not tuples, and with the exception of [19] limited to the propositional case. The application of the results obtained in this area to database queries is unclear.

The papers on *preference reasoning* [24,22,4] attempt to develop practical mechanisms for making inferences about preferences and solving decision or configuration problems similar to the one described in Example 4. A central notion there is that of *ceteris paribus* preference: preferring one outcome to another, all else being equal. Typically, the problems addressed in this work are propositional (or finite-domain). Such problems can be encoded in the relational data model and the inferences obtained by evaluating preference queries. A detailed study of such an approach remains still to be done. We note that the use of a full-fledged query language in this context makes it possible to formulate considerably more complex decision and configuration problems than before.

The work on *prioritized logic programming and nonmonotonic reasoning* [5,6,21] has potential applications to databases. However, like [11] it relies on specialized evaluation mechanisms.

8 Conclusions and future work

We have presented a framework for specifying preferences using logical formulas and its embedding into relational algebra. As the result, preference queries and complex decision problems involving preferences can be formulated in a simple and clean way.

Clearly, our framework is limited to applications that can be entirely modeled within the relational model of data. Here are several examples that do not quite fit in this paradigm:

- preferences defined between *sets* of elements;
- *heterogenous* preferences between tuples of different arity or type (how to say I prefer a meal without a wine to a meal with one in Example 4?);
- preferences requiring nondeterministic choice. We believe this is properly handled using a nondeterministic choice [9] or witness [1] operator.

In addition to addressing the above limitations, future work directions include:

- evaluation and optimization of preference queries;
- merging and propagation of preference relations;
- extrinsic preferences;
- defeasible and default preferences;
- preference elicitation.

Acknowledgments

This paper is dedicated to the memory of Javier Pinto whose premature death prevented him from participating in this research. The conversations with Svet Braynov, Jarek Gryz, Bharat Jayaraman, and Jorge Lobo, and the comments by the anonymous referees are gratefully acknowledged. Special thanks go to Agnieszka Grabska for her skeptical enthusiasm and timely feedback.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. R. Agrawal and E.L. Wimmers. A Framework for Expressing and Combining Preferences. In *ACM SIGMOD International Conference on Management of Data*, pages 297–306, 2000.
3. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *IEEE International Conference on Data Engineering*, pages 421–430, 2001.
4. C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with Conditional Ceteris Paribus Preference Statements. In *Symposium on Uncertainty in Artificial Intelligence*, 1999.
5. G. Brewka and T. Eiter. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.
6. J. P. Delgrande, T. Schaub, and H. Tompits. Logic Programs with Compiled Preferences. In *European Conference on Artificial Intelligence*, 2000.
7. P. Fishburn. Preference Structures and their Numerical Representations. *Theoretical Computer Science*, 217:359–383, 1999.

8. P.C. Fishburn. *Utility Theory for Decision Making*. Wiley & Sons, 1970.
9. F. Giannotti, S. Greco, D. Sacca, and C. Zaniolo. Programming with Non-determinism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 19(3-4), 1997.
10. K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Logic Programming. In *International Conference on Logic Programming*, pages 731–745, 1995.
11. K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Queries in Deductive Databases. *New Generation Computing*, pages 57–86, 2001.
12. S. Guo, W. Sun, and M.A. Weiss. Solving Satisfiability and Implication Problems in Database Systems. *ACM Transactions on Database Systems*, 21(2):270–293, 1996.
13. S. O. Hansson. Preference Logic. In D. Gabbay, editor, *Handbook of Philosophical Logic*, volume 8. 2001.
14. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multiparametric Ranked Queries. In *ACM SIGMOD International Conference on Management of Data*, pages 259–270, 2001.
15. R.G. Hughes. Rationality and Intransitive Preferences. *Analysis*, 40:132–134, 1980.
16. P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995.
17. W. Kiessling, S. Fischer, S. Holland, and T. Ehm. Design and Implementation of COSIMA – A Smart and Speaking E-sales Assistant. In *International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2001.
18. M. Lacroix and P. Lavency. Preferences: Putting More Knowledge Into Queries. In *International Conference on Very Large Data Bases*, pages 217–225, 1987.
19. S. M. Mantha. *First-Order Preference Theories and their Applications*. PhD thesis, University of Utah, 1991.
20. P. Z. Revesz. A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, 116:117–149, 1993.
21. C. Sakama and K. Inoue. Prioritized Logic Programming and its Application to Commonsense Reasoning. *Artificial Intelligence*, 123:185–222, 2000.
22. S-W. Tan and J. Pearl. Specification and Evaluation of Preferences under Uncertainty. In *International Conference on Principles of Knowledge Representation and Reasoning*, 1994.
23. G. H. von Wright. *The Logic of Preference*. Edinburgh University Press, 1963.
24. M. P. Wellman and J. Doyle. Preferential Semantics for Goals. In *National Conference on Artificial Intelligence*, pages 698–703, 1991.