Foundations of Preferences in Database Systems

Werner Kießling

Institute of Computer Science
University of Augsburg
D-86159 Augsburg, Germany
kiessling@informatik.uni-augsburg.de

Abstract

Personalization of e-services poses new challenges to database technology, demanding a powerful and flexible modeling technique for complex preferences. Preference queries have to be answered cooperatively by treating preferences as soft constraints, attempting a best possible match-making. We propose a strict partial order semantics for preferences, which closely matches people's intuition. A variety of natural and of sophisticated preferences are covered by this model. We show how to inductively construct complex preferences by means of various preference constructors. This model is the key to a new discipline called preference engineering and to a preference algebra. Given the Best-Matches-Only (BMO) query model we investigate how complex preference queries can be decomposed into simpler ones, preparing the ground for divide & conquer algorithms. Standard SQL and XPATH can be extended seamlessly by such preferences (presented in detail in the companion paper [15]). We believe that this model is appropriate to extend database technology towards effective support of personalization.

1. Introduction

Preferences are everywhere in all our daily and business lives. Recently they are catching wide-spread attention in the software community ([1]), in particular in terms of personalization for e-services. Thus it becomes also a challenge for database technology to adequately cope with the many sophisticated aspects of preferences. Personalization has different facets: There is the '*exact world*', where user wishes can be satisfied completely or not at all. In this scenario user options are restricted to a predefined set of fixed choices, e.g. for software configurations according to user profiles. Database queries in this context are characterized by *hard constraints*, delivering exactly the dream objects if they are there and otherwise reject the user's request. But there is also the *'real world'*, where personal preferences behave quite differently. Such preferences are understood in the sense of wishes: Wishes are free, but there is no guarantee that they can be satisfied at all times. In case of failure for a perfect match people are not always, but usually prepared to accept worse alternatives or to negotiate compromises. Thus preferences in the real world require a paradigm shift from exact matches towards a best possible *match-making*, i.e. preferences are to be treated as *soft constraints*. Moreover, preferences in the real world cannot be treated in isolation. Instead there may be multi-criteria decision situations where even multiple interested parties are involved, e.g. in e-shopping where e-customers and e-vendors have their own, maybe conflicting preferences. For a truly pervasive role of personalization these considerations suggest that database query languages should support both worlds. But whereas the exact-match paradigm been investigated in the database and Web context already by large amounts, leading to a bundle of successful technologies (e.g. SQL, E/R-modeling, XML), the paradigm of preference-driven choices in the real world is lagging behind.

Let us exemplify the unsatisfying state of the art by looking at those many SQL-based search engines of e-shops, which cannot cope adequately with real user preferences: All too often no or no reasonable answer is returned though one has tried hard filling out query forms to match one's personal preferences closely. Most probably, one has encountered answers before sounding like "no hotels, vehicles, flights, etc. could be found that matched your criteria; please try again with different choices". The case of repeatedly receiving *empty query results* turns out to be extremely disappointing to the user, and it is even

Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002

more harmful for the e-merchant. Dictating the user to leave some entries in the query form unspecified often leads to another unpleasant extreme: an *overloading* with lots of mostly *irrelevant information*. There have been some approaches to cope with these deficiencies, notably in the context of *cooperative database systems* ([9, 21]). There the technique of query relaxation has been studied in order to deal with the empty result problem. Since many decades preferences have also played a big role in the *economic* and *social sciences*, in particular for multi-attribute decision-making in *operations research* ([3, 12]). *Machine learning* and *knowledge discovery* ([19]) are further areas where preferences are under investigation. Each of these approaches and lines of research has explored some of the challenges put by preferences.

However, a comprehensive solution that paves the way for a smooth and efficient integration of preferences with database technology has not yet been published. We think that a viable preference model for database systems should meet the following list of desiderata:

(1) An intuitive semantics: Preferences must become first class citizens in the modeling process. This demands an intuitive understanding and declarative specification of preferences. A universal preference model should cover non-numerical as well as numerical ranking methods.
(2) A concise mathematical foundation: This requirement goes without saying, but of course the mathematical foundation must harmonize with the intuitive semantics.
(3) A constructive and extensible preference model: Complex preferences should be built up inductively from simpler ones using an extensible repertoire of preference constructors.
(4) Conflicts of preferences must not cause a system failure: Dynamic composition of complex preferences must be supported even in the presence of conflicts. A practical preference model should be able to live with conflicts, not to prohibit them or to fail if they occur.
(5) Declarative preference query languages: Matchmaking in the real world means bridging the gap between wishes and reality. This implies the need for a new query model other than the exact match model of declarative database query languages.

Preference SQL (for details see [15]) and Preference XPATH ([17]) are representatives of the latter. A novel PREFERRING-clause allows the user to conveniently specify soft constraints reflecting complex preferences. For motivation consider this Preference SQL query:

```
SELECT * FROM used_cars
WHERE make = 'Opel'
PREFERRING(category = 'cabrio' ELSE
           category = 'roadster')
  AND price AROUND 40000
  AND HIGHEST(power)
  AND mileage BETWEEN 20000, 30000;
```

The rest of this paper is organized as follows: Sect. 2 introduces the basics of preferences as strict partial orders. In Sect. 3 we present a powerful preference model as the key to preference engineering. Sect. 4 is concerned with the development of a preference algebra. Sect. 5 investigates issues of preference queries under the BMO query model and provides decomposition algorithms for complex preference queries. Practical aspects and related work are covered in Sect. 6. Sect. 7 summarizes our results and outlines ongoing work. All proofs are omitted here, but can be found in the extended version ([13]).

2. Preferences as strict partial orders

Preferences in the real world show up in different forms as everybody is aware of. A careful examination of their nature reveals that they share a fundamental common principle. Let's examine the daily life with its abundance of preferences coming from subjective feelings or other influences. In this familiar setting it turns out that people express their wishes frequently in terms like **"I like A better than B"**. This kind of preference modeling is universally applied and intuitively understood by everybody. In fact, every child learns to apply it from its earliest youth. Thinking of preferences in terms of 'better-than' has a very natural counterpart in mathematics: One can map them directly onto **strict partial orders**. People are intuitively used to deal with such preferences, in particular with those that are not expressed in terms of numerical scores. But there is also another part of real life which primarily is concerned with sophisticated economical or technical issues, where numbers do matter. One can easily recognize that numerical ranking can be subsumed under this semantics, too. Thus modeling preferences as strict partial orders holds great promises, which of course has been recognized at various opportunities in computer science and other disciplines before. Here this **key finding** receives our undivided attention.

A preference is formulated on a set of attribute names with an associated domain of values, which figuratively speaking is the 'realm of wishes'. When combining preferences P1 and P2, we decide that P1 and P2 *may overlap* on their attributes, allowing multiple preferences to coexist on the same attributes. This generality is due to our design principle that conflicts of preferences must be allowed in practice and must not be considered as a bug.

Let $A = \{A_1, A_2, \ldots, A_k\}$ denote a non-empty set of attribute names A_i associated with domains of values $\mathrm{dom}(A_i)$. Considering the order of components within a Cartesian product as irrelevant, we define:

$$\mathrm{dom}(A) = \mathrm{dom}(\{A_1, A_2, \ldots, A_k\}) := \mathrm{dom}(A_1) \times \mathrm{dom}(A_2) \times \ldots \times \mathrm{dom}(A_k)$$

Note that this definition includes, e.g., the following:

If $B = \{A_1, A_2\}$ and $C = \{A_2, A_3\}$,
then $\mathrm{dom}(B \cup C) = \mathrm{dom}(\{A_1, A_2\} \cup \{A_2, A_3\}) = \mathrm{dom}(A_1) \times \mathrm{dom}(A_2) \times \mathrm{dom}(A_3)$.

Definition 1 Preference P = (A, <P)

Given a set A of attribute names, a *preference P* is a strict partial order **P = (A, <P)**, where $<P \subseteq dom(A) \times dom(A)$.

Thus <P is irreflexive and transitive (which imply asymmetry). Important is this *intended interpretation:*
"**x <P y**" is interpreted as "**I like y better than x**".

Further: $range(<P) := \{x \in dom(A) \mid \exists y \in dom(A): (x, y) \in <P \text{ or } (y, x) \in <P\}$.

Since preferences reflect important aspects of the real world a good visual representation is essential.

Definition 2 Better-than graph, quality notions

In finite domains a preference P can be drawn as a directed acyclic graph G, called the *'better-than' graph* of P.[1] Given G for P we define the following *quality notions* between values x, y in G:

- x <P y, if y is predecessor of x in G.
- Values in G without a predecessor are *maximal* elements of P (**max(P)**), being at *level 1*.
- x is on *level j*, if the longest path from x to a maximal value has j-1 edges.
- If there is no directed path between x and y in G, then x and y are *unranked*.

Definition 3 Special cases of preferences

a) P = (A, <P) is a **chain** preference, if for all x, y ∈ dom(A), x ≠ y: x <P y ∨ y <P x
b) $\mathbf{S^{\leftrightarrow} = (S, \varnothing)}$ is called **anti-chain** preference, given any set of values S.
c) The **dual** preference $\mathbf{P^{\delta} = (A, <P^{\delta})}$ reverses the order on P: $x <P^{\partial} y$ iff y <P x
d) Given P = (A, <P), every $S \subseteq dom(A)$ induces a **subset** preference $\mathbf{P^{\subseteq} = (S, <P^{\subseteq})}$, if for any x, y ∈ S: $x <P^{\subseteq} y$ iff x <P y

Thus all values x of a chain preference P (also called total order) are ranked to all other values y. Any set S, including dom(A), can be converted into an anti-chain. Special subset preferences, called database preferences, will become important later on.

3. Preference engineering

Complex wishes are abundant in daily private and business life, even those concerning several attributes. Thus there is a high demand for a powerful and orthogonal framework that supports the accumulation of single preferences into more complex ones. We present an inductive approach towards constructing complex preferences. This model will be the key towards a systematic *preference engineering* and for a preference algebra.

[1] 'Better-than' graphs are also known as *Hasse diagrams* ([6]).

3.1. Inductive construction of preferences

The goal is to provide intuitive and convenient ways to inductively construct a preference P = (A, <P). To this end we specify P by a so-called *preference term* which fixes the attribute names A and the strict partial order <P. We distinguish between base preferences (our atomic preference terms) and compound preferences. Since each preference term represents a strict partial order (which becomes clear later on), we identify it with a preference P.

Definition 4 Preference term

Given preference terms P1 and P2, P is a *preference term* iff P is one of the following:

(1) Any **base** preference: $\mathbf{P := base pref_i}$.
(2) Any **subset** preference: $\mathbf{P := P1^{\subseteq}}$
(3) Any **dual** preference: $\mathbf{P := P1^{\partial}}$
(4) Any **complex** preference P gained by applying one of the following preference constructors:
   - **Accumulating** preference constructors:
     - Pareto accumulation: **P := P1 ⊗ P2**
     - Prioritized accumulation: **P := P1 & P2**
     - Numerical accumulation: $\mathbf{P := rank_F(P1, P2)}$
   - **Aggregating** preference constructors:
     - Intersection aggregation: **P := P1 ♦ P2**
     - Disjoint union aggregation: **P := P1 + P2**
     - Linear sum aggregation: **P := P1 ⊕ P2**

Both the set of base preferences and the set of complex preference constructors can be enlarged whenever the application domain at hand has a frequent demand.

3.2. Base preference constructors

Important from a preference engineering point is that we can provide base preference *constructors*, which in fact are *preference templates*, whose proper instantiations yield base preferences. Practical experiences from [15] showed that the following repertoire is highly valuable for constructing powerful personalized search engines.

Formally, a base preference constructor has one or more arguments, the first characterizing the attribute names A and the others the strict partial order <P, referring to A. We will provide both a *formal* and an *intuitive* definition together with a motivating example within a fictitious used_car application scenario.

3.2.1. Non-numerical base preferences

a) POS preference: POS(A, POS-set)
P is a *POS preference*, if:
$x <P y$ iff $x \notin$ POS-set ∧ $y \in$ POS-set
A desired value *should be* in a finite set of favorites POS-set ⊆ dom(A). If this infeasible, better than getting nothing any *other value* from dom(A) is acceptable. (This implies that all $v \in$ POS-set are maximal, all $v \notin$ POS-set are at level 2 and worse than all POS-set values.)

Used_car scenario:
`POS(transmission, {automatic})`

b) NEG preference: NEG(A, NEG-set)
P is a *NEG preference*, if:
$x <_P y$ iff $y \notin$ NEG-set $\wedge$ $x \in$ NEG-set
A desired value *should not be* any from a finite set NEG-set of dislikes. If this is infeasible, any disliked value is acceptable. (This implies that all $v \notin$ NEG-set are maximal, all $v \in$ NEG-set are at level 2 and worse than all maximal values.)
Used_car scenario: `NEG(make, {Ferrari})`

c) POS/NEG preference: POS/NEG(A, POS-set; NEG-set)
P is called *POS/NEG preference*, if:
$x <_P y$ iff $(x \in$ NEG-set $\wedge$ $y \notin$ NEG-set$) \vee$
$(x \notin$ NEG-set $\wedge$ $x \notin$ POS-set $\wedge$ $y \in$ POS-set)
A desired value *should be* one from a finite set of favorites. Otherwise it *should not be* any from a finite set of disjoint dislikes. If this is not feasible too, better than getting nothing any disliked value is acceptable.
Used_car scenario:
`POS/NEG(color, {yellow};{gray})`

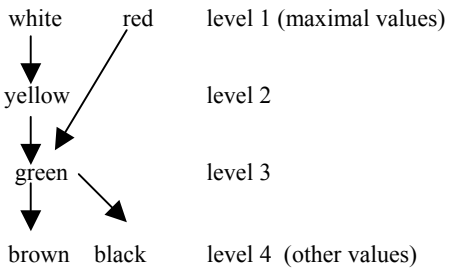
d) POS/POS preference: POS/POS(A, POS1-set; POS2-set)
P is called *POS/POS preference*, if:
$x <_P y$ iff $(x \in$ POS2-set $\wedge$ $y \in$ POS1-set$) \vee$
$(x \notin$ POS1-set $\wedge$ $x \notin$ POS2-set $\wedge$ $y \in$ POS2-set$) \vee$
$(x \notin$ POS1-set $\wedge$ $x \notin$ POS2-set $\wedge$ $y \in$ POS1-set)
A desired value *should be* amongst a finite set POS1-set. Otherwise it *should be* from a disjoint finite set of *alternatives* POS2-set. If this is not feasible too, better than getting nothing any other value is acceptable.
Used_car scenario:
`POS/POS(category,{cabrio};{roadster})`

Any finite preference can be "handcrafted" by explicitly enumerating 'better-than' relationships.

e) EXPLICIT preference: EXP(A, E-graph)
Let E-graph = $\{(val_1, val_2), \ldots\}$ represent a finite acyclic 'better-than' graph, V be the set of all val_i occurring in E-graph. A strict partial order $E = (V, <_E)$ is induced as follows:
- $(val_i, val_j) \in$ E-graph implies $val_i <_E val_j$
- $val_i <_E val_j \wedge val_j <_E val_k$ imply $val_i <_E val_k$

P is an *EXPLICIT preference*, if:
$x <_P y$ iff $x <_E y \vee (x \notin$ range$(<_E) \wedge y \in$ range$(<_E))$
Used_car scenario:
```
EXP(color, {(green, yellow),
       (green, red), (yellow, white)})
```

Given dom(Color) = {white, red, yellow, green, brown, black}, the 'better-than' graph is this:



3.2.2. Numerical base preferences

Now we focus on P = (A, <P), where dom(A) is some numerical data type, e.g. Decimal or Date, supporting a total comparison operator '<' and a subtraction operator '–'. Instead of the discrete *level* function above, we employ continuous *distance* functions defined on '<' and '–'.

a) AROUND preference: AROUND(A, z)
Given $z \in$ dom(A), for all $v \in$ dom(A) we define:
distance(v, z) := abs(v – z)
P is called *AROUND preference*, if:
$x <_P y$ iff distance(x, z) > distance(y, z)
The desired value *should be* z. If this is infeasible, values with *shortest distance apart* from z are acceptable.
Used_car scenario: `AROUND(price, 40000)`
Note that if distance(x, z) = distance(y, z) and $x \neq y$, then x and y are unranked.

b) BETWEEN preference: BETWEEN(A, [low, up])
Given [low, up] $\in$ dom(A) $\times$ dom(A), we define for all $v \in$ dom(A):
distance(v, [low, up]) :=
if $v \in$ [low, up] then 0 else
if $v <$ low then low – v else v – up
P is called *BETWEEN preference*, if: $x <_P y$ iff
distance(x, [low, up]) > distance(y, [low, up])
A desired value *should be between* the bounds of an interval. If this is infeasible, values with *shortest distance apart* from the interval boundaries will be acceptable.
Used_car scenario:
`BETWEEN(mileage, [20000, 30000])`

c) LOWEST, HIGHEST preference: LOWEST(A), HIGHEST(A)
P is called *LOWEST preference*, if: $x <_P y$ iff $x > y$
P is called *HIGHEST preference*, if: $x <_P y$ iff $x < y$
A desired value *should be* as *low (high)* as possible.
Used_car scenario: `HIGHEST(power)`
Note: LOWEST and HIGHEST preferences are chains.

Now let's revisit our introductory Preference SQL query. The preference term in the PREFERRING-clause specifies a Pareto accumulation as follows:

```
POS/POS(category,{cabrio};{roadster})⊗
AROUND(price,40000) ⊗ HIGHEST(power) ⊗
BETWEEN(mileage,[20000, 30000])
```

d) SCORE preference: SCORE(A, f)

We assume a *scoring function* f: dom(A) → ℝ. Let '<' be the familiar 'less-than' order on ℝ.
P is called *SCORE preference*, if for x, y ∈ dom(A):
x <P y iff f(x) < f(y)
In general no intuitive interpretation is available.

3.3. Complex preference constructors

The true power of preference modeling comes with the advent of complex preference constructors.

3.3.1. Accumulating preference constructors

Accumulating preference constructors ('⊗', '&', '$rank_F$') combine preferences which may come from one or several parties. The *Pareto-optimality principle* has been studied intensively for multi-attribute decision problems in the social and economic sciences. Here we define it for n = 2 preferences (generalizing it to n > 2 is obvious).

Definition 5 Pareto preference: P1⊗P2

P1 and P2 are considered as *equally important* preferences. In order for x = (x_1, x_2) to being better than y = (y_1, y_2), it is not tolerable that x is worse than y in any x_i:
Given P1 = (A1, <P1) and P2 = (A2, <P2), for x, y ∈ dom(A1) × dom(A2) we define:

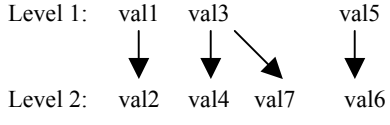
$$x <_{P1 \otimes P2} y \text{ iff } (x_1 <_{P1} y_1 \wedge (x_2 <_{P2} y_2 \vee x_2 = y_2)) \vee (x_2 <_{P2} y_2 \wedge (x_1 <_{P1} y_1 \vee x_1 = y_1))$$

P = (A1 ∪ A2, <P1⊗P2) is called *Pareto preference*[2]. The maximal values of P are the *Pareto-optimal set*.

Example 1 Pareto preference (disjoint attrib. names)

For dom(A1) = dom(A2) = dom(A3) = integer and
P1 := AROUND(A1, 0),
P2 := LOWEST(A2), P3 := HIGHEST(A3)
P4 = ({A1, A2, A3}, <P4) := (P1 ⊗ P2) ⊗ P3
let's study a subset preference of P4 for the following set:
R(A1, A2, A3) = {val1: (–5, 3, 4), val2: (–5, 4, 4),
val3: (5, 1, 8), val4: (5, 6, 6), val5: (–6, 0, 6),
val6: (–6, 0, 4), val7: (6, 2, 7)}

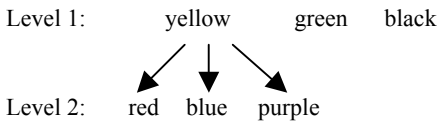
The 'better-than' graph of P4 for subset R can e.g. be obtained by performing exhaustive 'better-than' checks:



Thus the Pareto-optimal set is {val1, val3, val5}. Note that for each of P1, P2 and P3 at least one maximal value appears in the Pareto-optimal set: 5 and –5 for P1, 0 for P2 and 8 for P3. ☼

Example 2 Pareto preference (shared attribute names)

P5 := POS(Color, {green, yellow}),
P6 := NEG(Color, {red, green, blue, purple}),
P7 = (Color, <P7) := P5⊗P6,
S := {red, green, yellow, blue, black, purple}.
The 'better-than' graph of P7 for subset S is this:



Note that P5 and P6 agreed both on 'yellow' being maximal, whereas only P5 ranked 'green' as maximal and only P6 ranked 'black' as maximal. The result in P7 is a non-discriminating compromise of both views. ☼

Definition 6 Prioritized preference: P1&P2

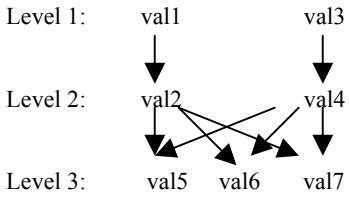
P1 is considered *more important* than P2; P2 is respected only where P1 does not mind:
Given P1 = (A1, <P1) and P2 = (A2, <P2), for x, y ∈ dom(A1) × dom(A2) we define:

$$x <_{P1\&P2} y \text{ iff } x_1 <_{P1} y_1 \vee (x_1 = y_1 \wedge x_2 <_{P2} y_2)$$

P = (A1∪A2, <P1&P2) is a *prioritized preference*.[3]

Example 3 Prioritization (disjoint attribute names)

Let's revisit Example 1, now studying:
P8 = ({A1, A2}, <P8) := P1&P2
The 'better-than' graph of P8 for subset R is this:



☼

Numerical preferences build on SCORE preferences. The individual scores are accumulated into an overall score by applying a multi-attribute combining function F. We define it for n = 2; generalizing it to n > 2 is obvious. An intuitive interpretation is not available in general.

Definition 7 Numerical preference: $rank_F$(P1, P2)

Given P1 = SCORE(A1, f1), P2 = SCORE(A2, f2) and a *combining function* F: ℝ × ℝ → ℝ, for x, y ∈ dom(A1) × dom(A2) we define: x <$rank_F$(P1, P2) y iff

$$F(f1(x_1), f2(x_2)) < F(f1(y_1), f2(y_2))$$

[2] Being a strict variant of the coordinate-wise order of Cartesian products ([6]), P is a strict partial order.

[3] It is a strict variant of the lexicographic order of Cartesian products ([6]), hence a strict partial order.

P = (A1∪A2, <rank$_F$(P1, P2)) is a *numerical preference*.

Note that rank$_F$ is *not* an orthogonal preference constructor like ⊗ or &. It can only be applied to SCORE preferences. But vice versa, numerical preferences can be used as input to all other preference constructors.

Example 4 Numerical preference (F as weighted sum)

P1 := SCORE(A1: Integer, f1), f1(x) := distance(x, 0)
P2 := SCORE(A2: Integer, f2), f2(x) := distance(x, –2)
P3 := rank$_F$(P1, P2), F(x1, x2) := x1 + 2 ∗ x2
R(A1, A2) := {val1: (–5, 3), val2: (–5, 4), val3: (5, 1), val4: (5, 6), val5: (–6, 0), val6 : (–6, 0)}

We evaluate f1 and f2 into a set Rankings, containing for each value of R its score vector for f1, f2 together with its combined F-ranking:

Rankings = {val1: ((5, 5), 15), val2: ((5, 6), 17), val3: ((5, 3), 11), val4: ((5, 8), 21), val5: ((6, 2), 10), val6: ((6, 2), 10)}

The 'better-than' graph of P3 for subset R is not a chain:

val4 → val2 → val1→ val3→ {val5, val6}

Observe that the maximal f1-value being 6 does not show up in the top performer val4, having scores (5, 8). In some sense this is like discriminating against P1. ☼

3.3.2. Aggregating preference constructors

Aggregating preference constructors (♦, +, ⊕) pursue a different, technical purpose. Intersection '♦' and disjoint union '+' *assemble* a preference P from *separate pieces* $P_1, P_2, \ldots, P_n$, all acting on the *same* set of attributes. Vice versa, we will see later on how complex preferences can be *decomposed* into '♦' and '+'.

Let's call P1 = (A1, <P1) and P2 = (A2, <P2) **disjoint** preferences, if range(<P1) ∩ range(<P2) = ∅.

Definition 8 Intersection, disjoint union preference

Assume P1 = (A, <P1) and P2 = (A, <P2).

a) **P = (A, <P1♦P2)** is an *intersection preference*, if:
x <P1♦P2 y iff x <P1 y ∧ x <P2 y

b) Given disjoint preferences P1 and P2, **P = (A, <P1+P2)** is called *disjoint union preference*, if:
x <P1+P2 y iff x <P1 y ∨ x <P2 y

Definition 9 Linear sum preference

Assume P1 = (A1, <P1), P2 = (A2, <P2) for single attributes A1 ≠ A2 and dom(A1) ∩ dom(A2) = ∅. Then P1 and P2 are disjoint preferences. For a new attribute name A let dom(A) := dom(A1) ∪ dom(A2).

Then **P = (A, <P1⊕P2)** is a *linear sum preference*, if:
x <P1⊕P2 y iff x <P1 y ∨ x <P2 y ∨ (x ∈ dom(A2) ∧ y ∈ dom(A1))

Linear sum '⊕' can be viewed as a convenient design and proof method for base preference constructors. With the proper notion of 'other-values' we can state:

A POS-preference is the linear sum of the anti-chain on the POS-set with the anti-chain on the other values:
POS = POS-set$^{↔}$ ⊕ other-values$^{↔}$
Similarly we observe that:
POS/NEG = (POS-set$^{↔}$ ⊕ other-values$^{↔}$) ⊕ NEG-set$^{↔}$
POS/POS = (POS1-set$^{↔}$ ⊕ POS2-set$^{↔}$) ⊕ other-values$^{↔}$
EXPLICIT = E ⊕ other-values$^{↔}$

At this point we can summarize all results stated so far as follows, referring back to Definition 4:

Proposition 1

Each preference term defines a strict partial order preference.

This theorem gives us the grand freedom to flexibly combine multiple preferences according to the specific requirements in an application situation. Let's coin the notion of *preference engineering* and demonstrate its potentials by a typical scenario from B2C e-commerce.

Example 5 Preference engineering scenario

Suppose that Julia wants to buy a used car for herself and her friend Leslie. Contemplating about her personal *customer preferences*, she comes up with this wish list:

P1 := POS/POS(category, {cabrio};{roadster})
P2 := POS(transmission,{automatic})
P3 := AROUND(horsepower, 100)
P4 := LOWEST(price), P5 := NEG(color, {gray})

Then Julia decides about the relative importance of these single preferences:

Q1 = ({color, category, transmission, horsepower, price}, <Q1) := P5 & ((P1 ⊗ P2 ⊗ P3) & P4)

Julia communicates her wish list Q1 to her car dealer Michael, who adds *domain knowledge* P6 about cars:

P6 := HIGHEST(year-of-construction)

Any piece of ontological knowledge can be entered at this stage. Because also *vendors* have their *preferences*, of course, Michael has another preference P7 of its own:

P7 := HIGHEST(commission)

Since Michael is a fair play guy, the query he is going to issue against his used car database is this:

Q2 = ({color, category, transmission, horsepower, price, year-of-construction, commission}, <Q2) := (Q1 & P6) & P7 = ((P5 & ((P1 ⊗ P2 ⊗ P3) & P4)) & P6) & P7

Note that when mixing customer with vendor preferences Michael had not to worry that potential conflicts would crash his used car e-shop. Just before Michael queries his car database against Q2 Leslie enters the scene. A discussion with Julia reveals that she has a different color taste:

P8 := POS/NEG(color, {blue};{gray, red})

In addition, Leslie convinces Julia that money should matter as much as color. Consequently, Q1 adapted to these new preferences reads as follows:

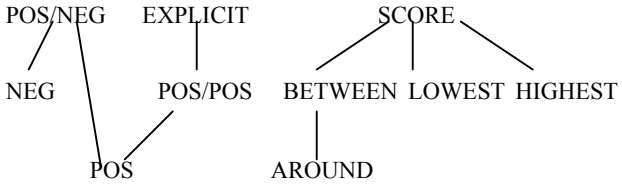
Q1* = ({color, category, transmission, horsepower, price}, <Q1) := (P5⊗P8⊗P4) & (P1⊗P2⊗P3)

Finally Michael poses Q2* … and the story might end that everybody is happy with the result. ☼

3.4. Preference hierarchies

Preference constructors C1 and C2 can be arranged in hierarchies. We call C1 a *preference sub-constructor* of C2 (**C1 < C2**), if the definition of C1 can be gained from the definition of C2 by some specializing constraints.

- Hierarchy of non-num. base preference constructors:
  - POS/POS < EXPLICIT, if E-graph = $(\text{POS1-set})^{\leftrightarrow} \oplus (\text{POS2-set})^{\leftrightarrow}$
  - POS < POS/POS, if POS2-set = ∅
  - POS < POS/NEG, if NEG-set = ∅
  - NEG < POS/NEG, if POS-set = ∅
- Hierarchy of numerical base preference constructors: ('N' means 'numeric')
  - BETWEEN < SCORE, if A is 'N' and f(x) = – distance(x, [low, up])
  - AROUND < BETWEEN, if low = up
  - HIGHEST < SCORE, if A is 'N' and f(x) = x
  - LOWEST < SCORE, if A is 'N' and f(x) = –x



- Hierarchy of complex preference constructors:
  - '♦' < '⊗'
  - Due to [5] not every preference constructor can be formulated as a sub-constructor of 'rank_F'.

Since we have specialization by constraints, sub-constructor hierarchies are taxonomic. Besides the usual advantages for object-oriented software engineering this also economizes proof efforts: Strict partial order semantics must be verified only for top-level preference constructors. Further we assume the principle of *constructor substitutability*, i.e. instead of a requested constructor also a sub-constructor can be supplied. For instance, rank_F(P1, P2) requires that P1 and P2 are SCORE preferences. Instead, we can e.g. also supply preferences P1 and P2 constructed by AROUND and HIGHEST, respectively.

4. A preference algebra

Hard constraints are formulated by first order logic formulas, which can be manipulated by Boolean algebra. On the other hand preferences, represented by preference terms, are used to express soft constraints. Therefore it is desirable to develop a preference algebra that can prove laws amongst preference terms. The subsequent studies will also strengthen our previous propositions about the intuitive semantics of preference constructors. First we need a notion of equivalence of preference terms.

Definition 10 Equivalence of preference terms

P1 = (A, <P1) and P2 = (A, <P2) are *equivalent* **(P1 ≡ P2)**, if for all x and y ∈ dom(A): x <P1 y iff x <P2 y

If P1 ≡ P2, then the preference terms P1 and P2 can be syntactically different, but the preferences represented by P1 and P2, resp., are actually the same.

4.1. A collection of algebraic laws

The next proposition is covered already by [6].

Proposition 2 Commutative and associative laws

a) $P1 \otimes P2 \equiv P2 \otimes P1$
   $(P1 \otimes P2) \otimes P3 \equiv P1 \otimes (P2 \otimes P3)$
b) $(P1 \,\&\, P2) \,\&\, P3 \equiv P1 \,\&\, (P2 \,\&\, P3)$
c) $P1 \blacklozenge P2 \equiv P2 \blacklozenge P1$
   $(P1 \blacklozenge P2) \blacklozenge P3 \equiv P1 \blacklozenge (P2 \blacklozenge P3)$
d) $P1 + P2 \equiv P2 + P1$
   $(P1 + P2) + P3 \equiv P1 + (P2 + P3)$
e) $(P1 \oplus P2) \oplus P3 \equiv P1 \oplus (P2 \oplus P3)$

Proposition 3 Further laws for preference terms

a) $(S^{\leftrightarrow})^{\partial} \equiv S^{\leftrightarrow}$ for any set S, $(P^{\partial})^{\partial} \equiv P$
b) $(P1 \oplus P2)^{\partial} \equiv P2^{\partial} \oplus P1^{\partial}$
c) $\text{HIGHEST} \equiv \text{LOWEST}^{\partial}$
d) $\text{POS}^{\partial} \equiv \text{NEG}$,
   $\text{NEG}^{\partial} \equiv \text{POS}$ if POS-set = NEG-set
e) $P \blacklozenge P \equiv P$
f) $P \blacklozenge P^{\delta} \equiv P \blacklozenge A^{\leftrightarrow} \equiv A^{\leftrightarrow}$ if P = (A, <P)
g) If P1 and P2 are chains, then
   P1 & P2 and P2 & P1 are chains.
h) $P \,\&\, P \equiv P \,\&\, P^{\partial} \equiv P$
i) $P \,\&\, A^{\leftrightarrow} \equiv P$ if P = (A, <P)
j) $A^{\leftrightarrow} \,\&\, P \equiv A^{\leftrightarrow}$ if P = (A, <P)
k) $P \otimes P \equiv P$, $A^{\leftrightarrow} \otimes P \equiv A^{\leftrightarrow} \,\&\, P$
l) $P \otimes A^{\leftrightarrow} \equiv P \otimes P^{\partial} \equiv A^{\leftrightarrow}$ if P = (A, <P)

These laws match our intuitive semantic expectations. E.g., let's pick $P \otimes P^{\partial} \equiv A^{\leftrightarrow}$: Since P and P^{∂} are equally important, in case of conflicts for values x and y none of them prevails, instead x and y remain unranked. Since P and P^{∂} are in conflict everywhere, the full domain becomes unranked, hence the anti-chain $A^{\leftrightarrow}$.

4.2. Decomposition of '&' and '⊗'

The following "discrimination" theorem reflects the intuitive semantics of prioritized accumulation.

Proposition 4 "Discrimination" theorem for P1&P2

(a) P1&P2 ≡ P1 if P1 = (A, <P1) and P2 = (A, <P2)
(b) $P1 \,\&\, P2 \equiv P1 + (A1^{\leftrightarrow} \,\&\, P2)$ if $A1 \cap A2 = \varnothing$

For shared attributes P2 is completely dominated by P1. In the disjoint case P1 is *more important than* P2, because P2 is respected only inside groups of equal A1-values, hence not disturbing P1's 'better-than' decisions on A1. In this intuitive sense P1 discriminates against P2. From a different angle, '&' can also be interpreted as a *conditional* preference: P2 becomes interesting only after P1 has happened.

Now we state the important "non-discrimination" theorem for Pareto accumulation, which likewise nicely supports our intuitive semantics for P = P1 ⊗ P2.

Proposition 5 "Non-discrimination" theorem

$$P1 \otimes P2 \equiv (P1 \,\&\, P2) \blacklozenge (P2 \,\&\, P1)$$

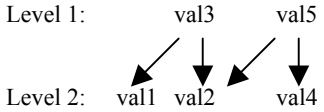
P1 and P2 are indeed treated *equally important* by '⊗', since both are given prime importance by '&'. Any arising conflict is resolved in a non-discriminating way by intersection '♦'. As a **corollary** we can state:

P1⊗P2 ≡ P1♦P2 if P1 = (A, <P1) and P2 = (A, <P2)

Thus '♦' is a preference sub-constructor of '⊗'.

Example 6 "Non-discrimination" theorem

P1: = LOWEST(price), P2 := LOWEST(mileage)
P := ({price, mileage}, <P1⊗P2)
We consider Car-DB from dom(Price) × dom(Mileage):
Car-DB = {val1: (40000, 15000),
val2: (35000, 30000), val3: (20000, 10000),
val4: (15000, 35000), val5: (15000, 30000)}
The 'better-than' graph of P = P1⊗P2 for subset Car-DB is this (obtainable e.g. by exhaustive better-than tests):



On the other hand let's determine (P1&P2) ♦ (P2&P1):
The 'better-than' graph of P' = P1&P2 for subset Car-DB yields a chain: val5 → val4 → val3 → val2 → val1
The corresponding 'better-than graph' of P'' = P2&P1 yields a chain: val3 → val1 → val5 → val2 → val4
The 'better-than' graph of P'♦ P'' for subset Car-DB is the same as for P1⊗P2. Note that it matches exactly the set of 'better-than' relationships shared by P' and P''. ☼

5. Evaluation of preference queries

In SQL databases life seems comparably simple. Queries against a relation R are formulated as hard constraints, leading to an all-or-nothing behavior: If the desired values are in R, you get exactly what you wanted, otherwise you get nothing at all. The latter deficiency is the *empty-result problem*. The *exact-match query model* can become a real nuisance in many e/m-commerce applications. The other extreme happens, if - being afraid of empty results - the query is built by disjunctive subqueries. Then one is frequently inundated with lots of irrelevant query results. This is the notorious *flooding-effect*.

The real world, where wishes are expressed as preferences, neither follows a simple all-or-nothing paradigm nor do people expect to be flooded with irrelevant values to choose from. Instead, a *cooperative answer semantics* is urgently needed. Whether preferences (i.e. wishes) can be satisfied and to what extent depends on the current status of the real world. Thus we have to perform a suitable *match-making* between wishes and reality. To this purpose we now define the so-called *BMO query model*.

5.1. The BMO query model

Preferences are defined in terms of values from dom(A), representing the realm of wishes. In database applications we assume that the real world is mapped into appropriate instances which we call database sets. A database set R may, e.g., be a view or a base relation in SQL or a DTD-instance in XML. Under the usual closed world assumption database sets capture the currently valid or accessible state of the real world. Thus they are subsets of our domains of values, hence they are subset preferences.

Consider a database set $R(B_1, B_2, \ldots, B_m)$. Given $A = \{A_1, A_2, \ldots, A_k\}$, where each A_j denotes an attribute B_i from R, let $R[A] := R[A_1, A_2, \ldots A_k]$ denote the projection π of R onto these k attributes.

Definition 11 Database preference P^R

Let's assume P = (A, <P), where $A = \{A_1, A_2, \ldots, A_k\}$.

a) Each $R[A] \subseteq dom(A)$ defines a subset preference, called a *database preference* and denoted by:
$P^R = (R[A], <P)$

b) Tuple $t \in R$ is a *perfect match* in a database set R, if:
$t[A] \in max(P) \wedge t[A] \in R$

Comparing max(P), i.e. the dream objects of P, with the set $max(P^R)$, i.e. the best objects available in the real world, then there might be no overlap. But if so, we have a perfect match between wishes and reality. If t is a perfect match for P in R, then $t[A] \in max(P^R)$. But the converse does not hold in general. Preference queries perform a match-making between the stated preferences (wishes) and the database preferences (reality).

Definition 12 Declarative semantics of σ[P](R)

Let's assume P = (A, <P) and a database preference P^R. We define a *preference query* **σ[P](R)** declaratively as follows: $\sigma[P](R) = \{t \in R \mid t[A] \in max(P^R)\}$

σ[P](R) evaluates P against a database set R by retrieving all maximal values from P^R. Note that not all of them are necessarily perfect matches of P. Thus the principle of *query relaxation* is implicit in above definition. Furthermore, any non-maximal values of P^R are excluded from the query result, hence can be considered as *discarded on*

the fly. In this sense all best matching tuples – and only those – are retrieved by a preference query. Therefore we coin the term **BMO query model** *("Best Matches Only")*.

Example 7 BMO query model

We revisit the sample explicit preference P of Sect. 3.2.1. e) and pose the query σ[P](R) for R(color) = {yellow, red, green, black}. The BMO result is: σ[P4](R) = {yellow, red}. Note that 'red' is a perfect match. ☼

As a straightforward, but important observation we state:
If P1 ≡ P2, then for all R: σ[P1](R) = σ[P2](R)

Besides preferences queries of the form σ[P](R) a variation will be needed frequently, which originates from an interesting interplay between grouping and anti-chains. Consider $\sigma[A^{\leftrightarrow}\&P](R)$, where P = (B, <P).
Since $x <_{A^{\leftrightarrow}\&P} y$ iff $x1 = y1 \wedge x2 <_P y2$, we have:
$t \in \sigma[A^{\leftrightarrow}\&P](R)$ iff
$\forall v[A, B] \in R[A, B]: \neg(t[A] = v[A] \wedge t[B] <_P v[B])$
In operational terms this characterizes a grouping of R by equal A-values, evaluating for each group G_i of tuples the preference query $\sigma[A^{\leftrightarrow}\&P](G_i)$. This motivates the following definition.

Definition 13 σ[P groupby A](R)

Given P = (B, <P) and a database preference P^R, a *preference query with grouping* **σ[P groupby A](R)** is defined as: $\sigma[P \text{ groupby } A](R) := \sigma[A^{\leftrightarrow}\&P](R)$

Compared to hard selection queries, preference queries deviate from the logics behind hard selections: Preference queries behave *non-monotonically*.

Example 8 Non-monotonicity of preference queries

For Cars(Fuel_Economy, Insurance_Rating, Nickname) let's consider:
P := HIGHEST(Fuel_Economy) ⊗
HIGHEST(Insurance_Rating),
We successively evaluate σ[P](Cars) for Cars as follows:
Cars = {(100, 3, frog), (50, 3, cat)}:
σ[P](Cars) = {(100, 3, frog)}
Cars = {(100, 3, frog), (50, 3, cat) (50, 10, shark)}:
σ[P](Cars) = {(100, 3, frog), (50, 10, shark)}
Cars = {(100, 3, frog), (50, 3, cat) (50, 10, shark), (100, 10, turtle)}:
σ[P](Cars) = {(100, 10, turtle)} ☼

Though we added more and more tuples, the results of our preference queries did not exhibit a similar behavior. Instead of adapting to the size of Cars, σ[P](Cars) *adapted to the quality* of data. The explanation is intuitive: Being *'better than' is not a property of a single value*, rather it concerns comparisons between pairs of values. Therefore it is sensitive (holistic) to the quality of a collection of values, and not to its sheer quantity. Thus "*quality instead of quantity*" is the name of the game for BMO.

5.2. Decomposition of '+' and '♦'-queries

A key challenge of preference query evaluation is to find efficient algorithms for complex preference constructors. For the scope of this paper we do *not* explicitly address efficiency issues, instead we provide fundamental decomposition results that can form the basis for divide-and-conquer approaches by preference query optimizers. Our main goal here is to decompose Pareto preferences into '+' and '♦', which in turn can be decomposed further.

Proposition 6 $\sigma[P1+P2](R) = \sigma[P1](R) \cap \sigma[P2](R)$

Next we need some technical definitions, given P = (A, <P) and a database preference P^R.

Definition 14 $\text{Nmax}(P^R)$, ${}_P\uparrow v$, $YY(P1, P2)^R$

a) The set of *non-maximal values* $\text{Nmax}(P^R)$ is defined as: $\text{Nmax}(P^R) := R[A] - \max(P^R)$
b) Given v ∈ dom(A), the *'better than' set* of v in P is defined as: ${}_P\uparrow v := \{w \in \text{dom}(A): v <_P w\}$
c) $YY(P1, P2)^R := \{t \in R : t[A] \in \text{Nmax}(P1^R) \cap \text{Nmax}(P2^R) \wedge {}_{P1}\uparrow t[A] \cap {}_{P2}\uparrow t[A] = \varnothing\}$

Proposition 7 Decomposition of '♦'-queries

$\sigma[P1 \blacklozenge P2](R) = \sigma[P1](R) \cup \sigma[P2](R) \cup YY(P1, P2)^R$

5.3. Decomposition of '&'-queries

Now we investigate σ[P1&P2](R). Since P1&P2 ≡ P1 for shared attributes (Proposition 4 a) we assume A1 ∩ A2 = ∅. The evaluation of prioritized preference queries can be done by *grouping*.

Proposition 8 Decomposition of '&'-queries

$\sigma[P1\&P2](R) = \sigma[P1](R) \cap \sigma[P2 \text{ groupby } A1](R)$

As a **corollary**, we obtain:
σ[P1&P2](R) = σ[P2](σ[P1](R)) , if P1 is a chain
Thus a **cascade** of preference queries is a special case of a prioritized preference query, if P1 is a chain.

Example 9 Decomposition of a prioritized query

We assume P1 := $\text{make}^{\leftrightarrow}$, P2 := AROUND(price, 40000) and a database set Cars(make, price, oid):
Cars = {(Audi, 40000, 1), (BMW, 35000, 2), (VW, 20000, 3), (BMW, 50000, 4)}
The informal query "For each make give me best offers with a price around 40000" translates into:
σ[P1&P2](Cars) =
σ[P1](Cars) ∩ σ[P2 groupby make](Cars) =
Cars ∩ {(Audi, 40000, 1), (BMW, 35000, 2), (VW, 20000, 3)} =
{(Audi, 40000,1), (BMW,35000,2), (VW,20000,3)} ☼

5.4. Decomposition of '⊗'-queries

Above results pave the ground for the main decomposition theorem for Pareto preference queries.

Proposition 9 Decomposition of '⊗'-queries

$$\sigma[P1 \otimes P2](R) = (\sigma[P1](R) \cap \sigma[P2 \text{ groupby } A1](R)) \cup (\sigma[P2](R) \cap \sigma[P1 \text{ groupby } A2](R)) \cup YY(P1\&P2, P2\&P1)^R$$

This theorem re-enforces also our claim that '⊗' treats P1 and P2 as equally important:

- The first and 2nd term contains all maximal values of $(P1\&P2)^R$ and $(P2\&P1)^R$, respectively.
- The 3rd term contains values that are neither maximal in $(P1\&P2)^R$ nor in $(P2\&P1)^R$.

Note that if P1 or P2 is a chain, then cascade of preferences can be applied, too.

Example 10 Evaluation of Pareto accumulation

Let P1 := LOWEST(A), P2 := HIGHEST(A) and R(A) := {3, 6, 9}. We compute σ[P1⊗P2](R). From the corollary to Proposition 5 and Proposition 3c, f) we can state:

$$\sigma[P1 \otimes P2](R) = \sigma[P1 \blacklozenge P2](R) = \sigma[P1 \blacklozenge P1^{\partial}](R) = \sigma[A^{\leftrightarrow}](R) = R$$

To countercheck, since both P1 and P2 are chains Proposition 9 specializes as follows:

$$\sigma[P1 \otimes P2](R) = \sigma[P2](\sigma[P1](R)) \cup \sigma[P1](\sigma[P2](R)) \cup YY(P1\&P2, P2\&P1)^R = \{3\} \cup \{9\} \cup YY(P1\&P2, P2\&P1)^R$$

We have: $Nmax((P1\&P2)^R) \cap Nmax((P2\&P1)^R) = \{6, 9\} \cap \{3, 6\} = \{6\}$

Since ${}_{P1\&P2}\uparrow 6 \cap {}_{P2\&P1}\uparrow 6 = \{3\} \cap \{9\} = \varnothing$, we get $YY(P1\&P2, P2\&P1)^R = \{6\}$

Thus: $\sigma[P1 \otimes P2](R) = \{3\} \cup \{9\} \cup \{6\} = R$ ☼

5.5. Filter effect of Pareto queries

Preference queries under BMO avoid both the empty-result and the flooding effect. On the other hand, search engines with an exact match query model struggle to combat those nuisances by offering patches like *parametric search*, which is a semi-automatic, repetitive attempt of query refinement, or by offering a so-called '*expert mode'*, being a Boolean query interface. However, this approach is known as inadequate for a long time ([24]).

We want to study more closely the filter effectiveness of preference queries under a BMO semantics. For P = (A, <P) let the *result size* of σ[P](R) be defined as:

$$size(P, R) := card(\pi_A(\sigma[P](R))) = card(max(P^R))$$

Definition 15 Strength of a preference filter

Given P1 = (A, <P1) and P2 = (A, <P2), P1 is a *stronger preference filter* than P2 (**P1→ P2**), if:

$$size(P1, R) \le size(P2, R).$$

Proposition 10 Filter strength of complex preferences

a) $P1+P2 \rightarrow P1,\ P1+P2 \rightarrow P2$
b) $P1 \rightarrow P1 \blacklozenge P2,\ P2 \rightarrow P1 \blacklozenge P2$
c) $P1\&P2 \rightarrow P1$
d) $P1\&P2 \rightarrow P1 \otimes P2,\ P2\&P1 \rightarrow P1 \otimes P2$

Let's interpret the filter effect of Pareto accumulation in a rough analogy to the Boolean 'AND/OR'-programming of search engines using an exact match query model. We can state:

$$P1 \otimes P2 \leftarrow P1\&P2 \rightarrow P1,\ P1 \otimes P2 \leftarrow P2\&P1 \rightarrow P2$$

From the point of view of P1 and P2, resp., forming P1&P2 and P2&P1 has stronger filter effects, hence resembling 'AND' operations in the exact match query model. Continuing to form P1⊗P2 has a weaker filter effect, resembling 'OR' operations. Since BMO automatically adapts to the quality of a database set R, as a net effect we get an *automatic* 'AND/OR'-like filter effect of Pareto accumulation. Thus BMO takes all this burden from the user by *automatically* finding best-matching answers.

6. Practical aspects and related work

Now we show how our complex preference model fits into database and Internet practice.

6.1. Integration into SQL and XML

6.1.1. Theoretical foundations

Declarative query languages under an exact query model, which includes object-relational SQL databases and XML databases, can be extended compatibly by strict partial order preferences under a BMO query model. The theory of *subsumption lattices* ([14, 20]), developed in the context of *Datalog_S*, provides the formal backbone, guaranteeing both the existence of a *model theory* and of a corresponding *fixpoint theory*.

6.1.2. Preference SQL

Preference SQL (for details see [15]), whose product release was available already in 1999, has been the first instance of an extension of SQL by preferences as strict partial orders. It implements a plug-and-go application integration by a clever rewriting of Preference SQL queries into SQL92-compliant code. Preference SQL is in commercial use as Preference Search cartridge for Intershop e-commerce platforms. The preference model implemented covers all previous base preference constructors, Pareto accumulation and cascading. Practical benchmarks showed that typical result sizes of Pareto preferences under BMO query semantics ranged from a few to a few dozens, which is exactly what's required in shopping situations ([16, 18]).

6.1.3. Preference XPATH

Preference XPATH ([17]) is a query language to build personalized query engines in XML environments. It can be applied in other XML key technologies like XSLT, Xpointer or Xquery. XPATH is extended as follows:
The production

```
LocationStep: axis nodetest predicate*
```

is upgraded as:

```
LocationStep: axis
        nodetest(predicate|preference)*
```

To delimit a hard selection (i.e. `predicate`) XPATH uses '[' and ']'. For soft selections (i.e. `preference`) '#[' and ']#' are used. Here is a sample query:

```
/CARS/CAR
  #[ (@fuel_economy)highest and
     (@mileage)lowest prior to
     (@color)in("black", "white")and
     (@price)around 10000 ]#
```

The equivalent preference term is as follows:

(HIGHEST(fuel_economy) ⊗ LOWEST(mileage)) &
(POS(color, {black, white}) ⊗ AROUND(price, 10000))

6.1.4. The 'SKYLINE OF' clause

The 'SKYLINE OF' clause for SQL proposed in [4] is a restricted form of Pareto accumulation

$P = P_1 \otimes P_2 \otimes \ldots \otimes P_n$,

where each P_i must be a LOWEST or HIGHEST preference, hence a chain. Efficient evaluation algorithms have been given in [4] and [22].

6.2. The ranked query model

Soft constraints in the form of numerical preferences are in use today in many database and information retrieval applications. In our model this amounts to preferences

$P = rank_F(SCORE(A_1, f_1), \ldots, SCORE(A_n, f_n))$.

Since $rank_F$ often yields chain preferences, a BMO query semantics would return exactly one best-matching object. This is a too small set to choose from in general. To get more alternative choices, the "*top-k*" query model is applied, returning k best-matching objects. This may amount to retrieve some non-maximal objects, too.

One use is in **multi-feature query engines**, e.g. to support queries by image content on color, texture or shape. There is already the SQL/MM proposal for incorporating ranked multi-feature queries into SQL. Efficient algorithms (see e.g. [10, 7]) can be used to speed up the computation of $rank_F$ under the "top-k" semantics. The PREFER system ([11]) is an instance of this ranked query model, too.

Another area are **full-text search engines**, where keywords can be understood as implicit SCORE preferences indicating their *relevance*. The combining function F for $rank_F$ is typically some scalar product using the cosine function, if the vector space model from information retrieval is used. SQL has been extended by text cartridges (extenders), implementing a top-k query model. The XXL prototype of [23] implements the top-k semantics in the XML context.

6.3. Other frameworks

The framework of *Agrawal / Wimmers* ([2]) falls somehow between the implementations of Preference SQL/XPATH and that of ranked query models. To express an 'I like x better than y' semantics SCORE preferences are used as base preference constructors, requiring that suitable numerical scores must be readily at hand. As a preference constructor so-called *combining forms* are provided, which have a closure property. In this way prioritization '&' and numerical ranking '$rank_F$' can be programmed. However, no declarative semantics of preference queries was given. Obviously the BMO query model can be a proper candidate, which can provide guidelines for an efficient implementation on top of a relational system (which was left as an open research issue).

The framework of *Chomicki* ([5]) emphasizes the view of preferences as strict partial orders, too, but defines preferences more generally as arbitrary logical formulas. He studies various classes of such formulas (intrinsic/extrinsic etc.) including prioritization as one preference constructor, but no Pareto preferences. The semantics of his *winnow*-operator coincides with our BMO query model. An embedding of preferences into relational query languages is proposed, but no practical implementations like Preference SQL / XPATH are given.

7. Summary and outlook

We presented a rich preference model tailored for database systems. Preferences as strict partial orders have an intuitive semantics; they may be subjective from daily life experiences, driven by personal intentions, or due to technical constraints. Our extensible preference model both unifies and extends existing approaches for non-numerical and numerical ranking and opens the door for a new discipline called preference engineering. Preferences as strict partial orders possess a Spartan formal basis being the key for a preference algebra, where many laws are valid that are valuable for preference query optimizers. We defined the declarative semantics of preference queries under the BMO query model, which can cope with the notorious empty-result and flooding problems of search engines. We also presented fundamental decomposition theorems for non-monotonic preference queries. Various portions of the presented preference model have already been prototyped or are in commercial use. Beyond the scope of this paper is the issue of efficiency of preference query evaluation, but [15] addresses several practical aspects.

There are new challenging research issues that can now be tackled. For *preference engineering* the tradeoffs between numerical ranking and non-numerical preferences are to be explored. From a user modeling perspective a plurality of preference constructors looks nice.

Whatever the choice is, a smooth integration of preferences into E/R- or UML modeling is highly desirable. On the other hand, a system implementor might prefer a lean choice, in the extreme case only $rank_F$ and SCORE preferences. However, the discussion on preference sub-constructor hierarchies showed that this is infeasible in general. Anyhow, due to the object-oriented taxonomy of preference hierarchies, more efficient sub-constructor implementations can be integrated easily on demand. Another area of increasing importance is *preference mining*. In particular for numerical preferences the issue of "where do all the numbers come from" matters a lot. The vastly increased preference modeling capabilities pose new challenges for mining algorithms from query log files, too.

Current preference research under the motto '*It's a Preference World*' at the University of Augsburg includes the following projects: P-NEWS (funded by the German Research Society DFG) applies preference engineering to a digital library application. COSIMA2 ([8], funded by the Bavarian Research Partnership FORSIP) investigates preference-based e-negotiation. Moreover, a preference query optimizer, exploiting our preference algebra and decomposition theorems, is being developed.

Acknowledgments:

I would like to thank T. Ehm, U. Güntzer and B. Möller for valuable suggestions to improve the technical presentation of this work.

Literature:

[1] Special issue of the Communications of the ACM on Personalization, vol. 43, Aug. 2000.

[2] R. Agrawal, E. L. Wimmers: *A Framework for Expressing and Combining Preferences*. Proc. ACM SIGMOD, May 2000, Dallas, pp. 297 - 306.

[3] K. Arrow: *Rational Choice Functions and Orderings.* Economica 26: pp. 121–127, 1959.

[4] S. Borzsonyi, D. Kossmann, K. Stocker: *The Skyline Operator*. Proc. 17th Intern. Conf. On Data Engineering, Heidelberg, Germany, April 2001.

[5] J. Chomicki: *Querying with Intrinsic Preferences*. Proc. Intern. Conf. on Advances in Database Technology (EDBT), Prague, March 2002, pp. 34-51.

[6] B.A. Davey, H.A. Priestley: *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge University press, 1990.

[7] R. Fagin, A. Lotem, M. Naor: *Optimal Aggregation Algorithms for Middleware*. Proc. ACM PODS, Santa Barbara, May 2001, pp.102 – 113.

[8] S. Fischer, W. Kießling, S. Holland, M. Fleder: *The COSIMA Prototype for Multi-Objective Bargaining,* First Intern. Joint Conf. on Autonomous Agents & Multiagent Systems, Bologna, July 2002.

[9] T. Gaasterland, J. Lobo: *Qualified Answers that Reflect User Needs and Preferences*. Proc. VLDB 1994, Santiago de Chile.

[10] U. Güntzer, W.-T. Balke, W. Kießling: *Optimizing Multi-Feature Queries for Image Databases*. Proc. VLDB 2000, Cairo, Egypt, Sept. 2000, pp. 419-428.

[11] V. Hristidis, N. Koudas, Y. Papakonstantinou : *PREFER : A System for the Efficient Execution of Multi-parametric Ranked Queries*. Proc. ACM SIGMOD, May 2001, Santa Barbara, pp. 259 - 269.

[12] R. Keeney, H. Raiffa: *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press, UK, 1993.

[13] W. Kießling: *Foundations of Preferences in Database Systems*. Technical Report 2001-8, Institute of Computer Science, Univ. of Augsburg, Oct. 2001. (www.Informatik.Uni-Augsburg.de/forschung/techBerichte/reports/2001-8.pdf)

[14] W. Kießling, U. Güntzer: *Database Reasoning - A Deductive Framework for Solving Large and Complex Problems by means of Subsumption*. Proc. 3rd Workshop on Information Systems and Artificial Intelligence, LNCS 777, pp. 118-138, Hamburg, 1994.

[15] W. Kießling, G. Köstler: *Preference SQL – Design, Implementation, Experiences*. Proc. 28th Intern. Conf. on Very Large Databases (VLDB), Hong Kong, China, Aug. 2002, this volume.

[16] W. Kießling, S. Fischer, S. Holland, T. Ehm: *Design and Implementation of COSIMA - A Smart and Speaking E-Sales Assistant*. Proc. 3rd Intern. Workshop on Advanced Issues of E-Commerce and Web-Based Inform. Syst., pp. 21-30, San Jose, June 2001.

[17] W. Kießling, B. Hafenrichter, S. Fischer, S. Holland: *Preference XPATH: A Query Language for E-Commerce*. Proc. 5th Intern. Konf. für Wirtschaftsinformatik, Augsburg, pp. 425-440, Sept. 2001.

[18] W. Kießling, S. Holland, S. Fischer, T. Ehm: *COSIMA - Your Smart, Speaking E-Salesperson*. ACM SIGMOD, Santa Barbara, May 2001, demo, p. 600.

[19] A. Kiss, J. Quinqueton: *Multiagent Cooperative Learning of User Preferences*, 5th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD), Freiburg, Sept. 2001.

[20] G. Köstler, W. Kießling, H. Thöne, U. Güntzer: *Fixpoint Iteration with Subsumption in Deductive Databases*. Journal of Intelligent Information Systems, Vol. 4, pp. 123-148, Boston, USA, 1995.

[21] J. Minker: *An Overview of Cooperative Answering in Databases*. Proc. 3rd Intern. Conf. on Flexible Query Answering Systems, Springer LNCS 1495, pp. 282-285, Roskilde, Denmark, 1998.

[22] K.-L. Tan, P.-K. Eng, B. C. Ooi: *Efficient Progressive Skyline Computation*. Proc. 27th Intern. Conf. on Very Large Datab., pp. 301-310, Rome, Sept. 2001.

[23] A. Theobald, G. Weikum: *Adding Relevance to XML.* Proc. of the 3rd Intern. Workshop on the Web and Databases, LNCS, Springer, 2000.

[24] J. Verhoeff, W. Goffmann, J. Belzer: Inefficiency of the Use of Boolean Functions for Information Retrieval Systems, CACM, Dec. 1961, Vol. 4, No. 2.