# Universität Augsburg

# Foundations of Preferences in Database Systems

**Werner Kießling**

Report 2001-8 Oktober 2001

Institut für Informatik
D-86135 Augsburg

# Foundations of Preferences in Database Systems

**Werner Kießling**

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


## Abstract

Personalization of e-services poses new challenges to database technology. In particular, a powerful and flexible modeling technique is needed for complex preferences, which may even come from several parties with different intentions. Preference queries against a database 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 broad variety of natural preferences and of sophisticated preferences using ranked scores are covered by this model. Moreover, we show how to inductively construct complex preferences from base preferences by means of various preference constructors including Pareto accumulation. This preference model is the key to a new discipline called preference engineering and to a preference algebra. We present a collection of laws, including an intuitive non-discrimination theorem for Pareto preferences. Given the Best-Matches-Only query model we investigate how complex preference queries can be decomposed into simpler ones, preparing the ground for divide & conquer algorithms. We succeed to verify interesting adaptive filter effects of preference queries. Standard database query languages can be extended seamlessly by such preferences as exemplified by Preference SQL and Preference XPATH. In summary we believe that this preference model, featuring an algebraic foundation that matches intuition, is appropriate to extend database technology by preferences as soft constraints. Building efficient preference query optimizers, which can cope with the intrinsic non-monotonic nature of preference queries, investigations on how to e-negotiate in this preference model and a systematic approach to preference engineering are now feasible steps towards advanced database support for the ubiquitous real world phenomenon of preferences.

## 1 Introduction

Preferences are everywhere in all our daily and business lives. Recently they are catching wide-spread attention in the software community ([ACM00]), in particular in terms of personalization for B2C or B2B 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 pre-defined 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 best possible *match-making*, which means that 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 more harmful for the e-merchant. Dictating the user to leave some entries in the query form unspecified often leads to the other 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* ([Mot88, GaL94, CYC96, Min98]). 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* ([Arr59, KeR93, BLL01]). *Machine learning* and *knowledge discovery* ([KiQ01]) 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, covering a wide spectrum of applications:* Preferences must be included as first class citizens into 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, and it should smoothly integrate with hard constraints from the exact world.

*(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:* Elevating preferences to the rank of first class citizens for application modeling requires that a rich preference model is supported. Complex preferences should be built up inductively from simpler preferences 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:* Match-making 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 query languages like SQL or XPATH. A smooth integration and an efficient implementation are prerequisites for its widespread acceptance.

The rest of this paper is organized as follows: Section 2 introduces the basics of preferences as strict partial orders. In section 3 we present a powerful preference model as the key to preference engineering. Section 4 is concerned with the development of a preference algebra. Section 5 investigates issues of preference queries under the BMO query model and provides decomposition algorithms for complex preference queries. Practical aspects of preference query languages are covered in section 6. Finally section 7 summarizes our results and outlines ongoing and future work.

## 2 Preferences as strict partial orders

Preferences in the real world show up in quite different forms as everybody is aware of. However, a careful examination of their vary nature reveals that they share a fundamental common principle. Let's examine the domain of daily life with its abundance of preferences that may come from subjective feelings or other intuitive 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 such real life preferences straightforwardly 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 the familiar numerical ranking can be subsumed under this semantics. Therefore modeling preferences as strict partial orders holds great promises, which of course has been recognized at various opportunities and situations in computer science and other disciplines before. Here this key finding receives our undivided attention.

A preference is formulated as strict partial order 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 into another preference P, 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 denote a non-empty set of *attribute names,* where each single attribute $A_i$ has an associated domain of values $dom(A_i)$: $A = \{A_1\text{: data\_type}_1, A_2\text{: data\_type}_2, \ldots, A_k\text{: data\_type}_k\}$

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

For brevity we often omit the data types; if A has only one element, we omit set notation. The order of components within the Cartesian product is considered irrelevant. Following above design decision this definition includes, e.g., the following: If $B = \{A_1, A_2\}$ and $C = \{A_2, A_3\}$, then $dom(B \cup C) = dom(\{A_1, A_2\} \cup \{A_2, A_3\}) = dom(A_1) \times dom(A_2) \times dom(A_3)$.

**Definition 1** **Preference P = (A, <P)**

A **preference P** is a strict partial order **P = (A, <P)**, where A is a set of attribute names and $<P \subseteq \mathrm{dom}(A) \times \mathrm{dom}(A)$. Thus <P is irreflexive and transitive (which imply asymmetry).[1]

Thus for all $x, y, z \in \mathrm{dom}(A)$ we have: [irreflexivity] $\neg(x <P\ x)$

[transitivity] $x <P\ y \;\wedge\; y <P\ z$ imply $x <P\ z$

Important is this *intended interpretation:* "**x <P y**" is interpreted as "**I like y better than x**"

A distinctive feature of partial orders are unranked values, i.e. values x and y such that $\neg(x <P\ y) \;\wedge\; \neg(y <P\ x)$ holds. 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. In mathematical terms 'better-than' graphs are known as *Hasse diagrams* ([DaP90]).

Given a 'better-than' graph G for P we define the following *quality notions* between values x, y in G:

- x <P y (i.e. y is **better than** x), if y is predecessor of x in G.
- Values in G without a predecessor are **maximal** elements of P, being at **level 1**.
- Values in G without a successor are **minimal** elements of P.
- 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**.

In numerical domains we will use a continuous **distance** function (see examples later on) instead of the discrete **level** function to describe quality notions. Here is a formal definition of maximal values, given P = (A, <P):

$$\mathbf{max(P)} := \{v \in \mathrm{dom}(A) \mid v \text{ is maximal in } P\} = \{v \in \mathrm{dom}(A) \mid \neg(\exists\ w \in \mathrm{dom}(A):\ v <P\ w\}$$

**Definition 3** **Special cases of preferences**

a) **Chain** preference: P = (A, <P) is called chain preference, if for all $x, y \in \mathrm{dom}(A)$, $x \neq y$: $x <P\ y \;\vee\; y <P\ x$

b) **Anti-chain** preference: $\mathbf{S^{\leftrightarrow} = (S, \emptyset)}$ is called anti-chain preference, given any set of values S.

c) **Dual** preference: The dual preference $\mathbf{P^{\delta} = (A, <P^{\delta})}$ reverses the order on P: $x <P^{\partial}\ y$ iff $y <P\ x$

d) **Subset** preference: Given P = (A, <P), every subset $S \subseteq \mathrm{dom}(A)$ induces a preference $\mathbf{P^{\subseteq} = (S, <P^{\subseteq})}$ called subset preference of P, if for any $x, y \in S$: $x <P^{\subseteq}\ y$ 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) for an attribute A, can be converted into an anti-chain. Special subset preferences, called database preferences, will become important later on, when we discuss the issue of preference queries.

**Definition 4** **range(<P), disjoint preferences**

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

P1 = (A1, <P1) and P2 = (A2, <P2) are called *disjoint* preferences, if: $\mathrm{range}(<P1) \cap \mathrm{range}(<P2) = \emptyset$

---

[1] Some people prefer to deal with non-strict partial orders ≤P. Mathematically, any strict partial order can be translated into its non-strict form in a canonical way ([DaP90]).

# 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. This accumulation should follow some general principles that are present in real life and have an intuitive semantics. We present an inductive approach towards constructing complex preferences. This preference model will enable us to perform a systematic *preference engineering*. It will likewise define the formal basis for the preference algebra introduced later on.

## 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 we will prove later on), we identify it with a preference P.

**Definition 5 Preference term**

P is a **preference term** if and only if P is one of the following:

(1) Any *base* preference $\text{basepref}_i$.

(2) Any *subset* preference of a preference P1: **P := P1$^{\subseteq}$**

(3) Any *dual* preference of a preference P1: **P := P1$^{\partial}$**

(4) Any *complex* preference P gained by applying one of the following preference constructors to given preferences P1 and P2:

- *Accumulating* preference constructors:
  - Pareto accumulation: **P := P1 ⊗ P2**
  - Prioritized accumulation: **P := P1 & P2**
  - Numerical accumulation: **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**

We assume a finite set a base preferences {$\text{basepref}_1$, $\text{basepref}_2$, …}, where each $\text{basepref}_i$ is assured to represent a strict partial order. Each of the stated preference constructors will be defined subsequently and will be proven to be closed under strict partial order semantics. Note that 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 for it.

## 3.2 Base preference constructors

Important from a preference engineering point of view is that we can provide *base preference constructors*, which can be considered as *preference templates* whose proper instantiation yields a base preference. Practical experiences ([KiK01]) showed that for e-shopping applications the following repertoire is highly valuable for constructing powerful personalized search engines. Formally, a base preference constructor has two arguments, the first characterizing the attribute names A and the second the strict partial order $<_P$. In the subsequent definitions we provide both an intuitive and a formal definition, distinguishing between *non-numerical* (POS, NEG, POS/NEG, POS/POS, EXPLICIT) and *numerical* base preference constructors (AROUND, BETWEEN, LOWEST, HIGHEST, SCORE).

### 3.2.1 Non-numerical base preferences

**Definition 6 Non-numerical base preference constructors**

**a) POS preference: P := POS(A, POS-set{$v_1, \ldots, v_m$})**

*Intuitively*: A desired value *should be* one from a set of favorites $v_1, \ldots, v_m \in$ dom(A), called *positive* values. If this is not feasible, better than getting nothing any *other value* from dom(A) is acceptable.

*Formally*: Let POS-set $\subseteq$ dom(A) be finite. P is a **POS preference**, if: $x <_P y$ iff $x \notin$ POS-set $\wedge$ $y \in$ POS-set

All $v \in$ POS-set are maximal, all $v \notin$ POS-set are at level 2 and worse than all POS-set values.

**b) NEG preference: P := NEG(A, NEG-set{$v_1, \ldots, v_m$})**

*Intuitively*: A desired value *should not be* any from a set of dislikes $v_1, \ldots, v_m \in$ dom(A), called *negative* values. If this is not feasible, better than getting nothing any disliked value is acceptable.

*Formally*: Let NEG-set $\subseteq$ dom(A) be finite. P is a **NEG preference**, if: $x <_P y$ iff $y \notin$ NEG-set $\wedge$ $x \in$ NEG-set

All $v \notin$ NEG-set are maximal, all $v \in$ NEG-set are on level 2 and worse than all maximal values.

**c) POS/NEG preference: P := POS/NEG(A, POS-set{$v_1, \ldots, v_m$}; NEG-set{$v_{m+1}, \ldots, v_{m+n}$})**

*Intuitively*: A desired value *should be* one from a set of favorites. Otherwise it *should not be* any from a set of dislikes. If this is not feasible too, better than getting nothing any disliked value is acceptable.

*Formally:* Let POS-set, NEG-set $\subseteq$ dom(A) be finite and disjoint. P is called **POS/NEG preference**, if:

$$x <_P y \text{ iff } (x \in \text{NEG-set} \wedge y \notin \text{NEG-set}) \vee (x \notin \text{NEG-set} \wedge x \notin \text{POS-set} \wedge y \in \text{POS-set})$$

All $v \in$ POS-set are maximal, all $v \in$ NEG-set are on level 3, all others are on level 2. All maximal values are better than all level 2 values which are better than all level 3 values.

**d) POS/POS preference: P := POS/POS(A, POS1-set{$v_1, \ldots, v_m$}; POS2-set{$v_{m+1}, \ldots, v_{m+n}$})**

*Intuitively*: A desired value *should be* one from a set of favorites. Otherwise it *should be* from a set of positive *alternatives*. If this is not feasible too, better than getting nothing any other value is acceptable.

*Formally*: Let POS1-set, POS2-set $\subseteq$ dom(A) be finite and disjoint. POS1-set are the favorite values, POS2-set are the second-best alternatives. P is called **POS/POS preference**, if:

$$\begin{aligned} x <_P y \text{ iff } & (x \in \text{POS2-set} \wedge y \in \text{POS1-set}) \vee \\ & (x \notin \text{POS1-set} \wedge x \notin \text{POS2-set} \wedge y \in \text{POS2-set}) \vee \\ & (x \notin \text{POS1-set} \wedge x \notin \text{POS2-set} \wedge y \in \text{POS1-set}) \end{aligned}$$

All $v \in$ POS1-set are maximal, all $y \in$ POS2-set are on level 2, all others are on level 3. All POS1-set values are better than all POS2-set values which are better than all other values.

**e) EXPLICIT preference: P := EXPLICIT(A, EXPLICIT-graph{($val_1$, $val_2$), … })**

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

*Formally*: Let EXPLICIT-graph = {($val_1$, $val_2$), … } represent a finite acyclic 'better-than' graph, where $val_i \in$ dom(A). Let V be the set of all $val_i$ occurring in EXPLICIT-graph. Then a strict partial order E = (V, <E) is induced as follows:

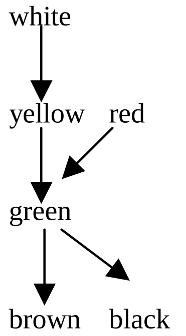
- ($val_i$, $val_j$) $\in$ EXPLICIT-graph implies $val_i$ <E $val_j$
- $val_i$ <E $val_j$ $\wedge$ $val_j$ <E $val_k$ imply $val_i$ <E $val_k$

P is an **EXPLICIT preference**, if: x <P y iff x <E y $\vee$ (x $\notin$ range(<E) $\wedge$ y $\in$ range(<E))

Note that all values in EXPLICIT-graph are better than all other values in dom(A).

**Example 1 Construction of base preferences**

- P = (Transmission, <P) := POS(Transmission, POS-SET{automatic})
- P = (Color, <P) := POS/NEG(Color, POS-set{yellow}; NEG-set{gray})
- P = (Category, <P) := POS/POS(Category, POS-set1{cabriolet}; POS-set2{roadster})
- P = (Color, <P) := EXPLICIT(Color, EXPLICIT-graph{(green, yellow), (green, red), (yellow, white)}) Given dom(Color) = {white, red, yellow, green, brown, black} the 'better-than' graph of P is this:




Thus white and red are maximal at level 1, yellow is at level 2, green is at level 3 and the other values brown and black are minimal at level 4.

☺

### 3.2.2 Numerical base preferences

Now we focus on preferences P = (A, <P), where dom(A) is some numerical data type, e.g. Real or Decimal. Then a *total comparison operator* **'<'** and the subtraction operator '–' are predefined on dom(A). Instead of the discrete **level** function above, we now employ a continuous **distance** function working on '<' and '–'.

**Definition 7 Numerical base preference constructors**

**a) AROUND preference: P := AROUND(A, z)**

*Intuitively*: A desired value *should be* an explicitly stated value z. If this is not feasible, values with *shortest distance apart* from z will be acceptable.

*Formally*: Given a value 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)

Note that if distance(x, z) = distance(y, z) and x $\neq$ y, then x and y are unranked. AROUND preferences (and the following base preferences) are also applicable to other ordered SQL types like Date.

**b) BETWEEN preference: P := BETWEEN(A, [low, up])**

*Intuitively*: A desired value *should be between* the bounds of an explicitly stated interval. If this is not feasible, values with *shortest distance apart* from the interval boundaries will be acceptable.

*Formally*: Given an interval $[low, up] \in dom(A) \times dom(A)$, $low \leq up$, for all $v \in dom(A)$ we define:

$$distance(v, [low, up]) := \text{ if } v \in [low, up] \text{ then } 0 \text{ else if } v < low \text{ then } low - v \text{ else } v - up$$

P is called **BETWEEN preference**, if: $x <_P y$ iff $distance(x, [low, up]) > distance(y, [low, up])$

Note that if $distance(x, [low, up]) = distance(y, [low, up])$ and $x \neq y$, then x and y are unranked.

**c) LOWEST, HIGHEST preference: P := LOWEST(A) , P = HIGHEST(A)**

*Intuitively*: A desired value *should be* as *low (high)* as possible.

*Formally :* P is called **LOWEST preference**, if: $x <_P y$ iff $x > y$

P is called **HIGHEST preference**, if: $x <_P y$ iff $x < y$

LOWEST and HIGHEST preferences are chains.

**d) SCORE preference: P := SCORE(A, f)**

*Intuitively*: Not available in general.

*Formally*: We assume a *scoring function* $f: dom(A) \rightarrow \mathbb{R}$. Let '<' be the familiar 'less-than' order on $\mathbb{R}$. Then

P is called **SCORE preference**, if for $x, y \in dom(A)$: $x <_P y$ iff $f(x) < f(y)$

Note that P need not be a chain, if the scoring function f is not a one-to-one mapping.

## 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 combine preferences which may come from one or several parties. We consider Pareto accumulation '⊗', prioritized accumulation '&' and numerical accumulation 'rank(F)').

The *Pareto-optimality principle* has been applied and studied intensively for decades for multi-attribute decision problems in the social and economic sciences. In our context we define it for n = 2 preferences as follows (a generalization to n > 2 is straightforward).

**Definition 8 Pareto preference: P:= P1⊗P2**

*Intuitively*: P1 and P2 are considered as **equally important preferences**. In order for v = (v1, v2) to being better than w = (w1, w2), it is not tolerable that v is worse than w in any component value.

*Formally:* We assume two preferences $P1 = (A1, <_{P1})$ and $P2 = (A2, <_{P2})$. For $x = (x1, x2)$ and $y = (y1, y2) \in dom(A1) \times dom(A2)$ we define:

$$x <_{P1\otimes P2} y \text{ iff } (x1 <_{P1} y1 \wedge (x2 <_{P2} y2 \vee x2 = y2)) \vee$$
$$(x2 <_{P2} y2 \wedge (x1 <_{P1} y1 \vee x1 = y1))$$

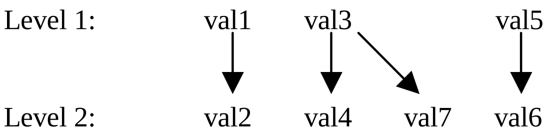
**$P = (A1 \cup A2, <_{P1\otimes P2})$** is called **Pareto preference**. The maximal values of P are the **Pareto-optimal set**. Being a strict variant of the coordinate-wise order of cartesian products ([DaP90]), P is a strict partial order.

**Example 2** **Pareto preference (disjoint attribute names)**

Given dom(A1) = dom(A2) = dom(A3) = integer, we consider P1 := AROUND(A1, 0), P2 := LOWEST(A2), P3 := HIGHEST(A3) and a Pareto preference P4 = ({A1, A2, A3}, <P4) := (P1 ⊗ P2) ⊗ P3. Let's study a subset preference of P4 for the following set R of values:

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)}

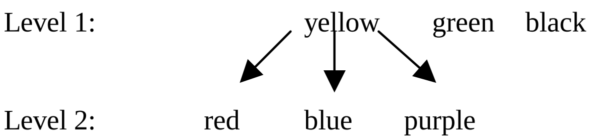
The 'better-than' graph of P4 for subset R can e.g. be obtained by performing exhaustive 'better-than' checks:




Thus in this case 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 3** **Pareto preference (shared attribute names)**

Let's assume P5 := POS(Color, POS-set{green, yellow}), P6 := NEG(Color, NEG-set{red, green, blue, purple}), a Pareto preference P7 = (Color, <P7) := P5⊗P6 and a set of colors S := {red, green, yellow, blue, black, purple}. The 'better-than' graph of P7 for subset S looks as follows:




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 9** **Prioritized preference: P:= P1&P2**

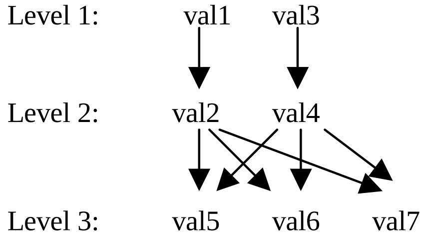
*Intuitively*: P1 is considered **more important** than P2. There is no compromise by P1; P2 is respected only where P1 does not mind.

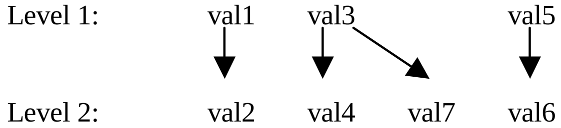
*Formally:* We assume two preferences P1 = (A1, <P1) and P2 = (A2, <P2). For x = (x1, x2) and y = (y1, y2) $\in$ dom(A1) $\times$ dom(A2) we define: x <P1&P2 y iff x1 <P1 y1 $\vee$ (x1 = y1 $\wedge$ x2 <P2 y2)

**P = (A1 $\cup$ A2, <P1&P2)** is called **prioritized preference**. It is a strict variant of the lexicographic order of cartesian products ([DaP90]), hence a strict partial order.

**Example 4** **Prioritized accumulation (disjoint attribute names)**

Let's revisit Example 2 from Pareto accumulation, now introducing two prioritized preferences P8 = ({A1, A2}, <P8) := P1&P2 and P9 = ({A1, A2, A3}, <P9) := (P1⊗P2) & P3. The 'better-than' graphs of P8 (left) and of P9 (right) for subset R look as follows:




☺

Numerical accumulation builds on SCORE preferences $P_1, P_2, \ldots, P_n$. The individual scores are accumulated into an overall score by applying a multi-attribute combining function F. We give its formal definition for n = 2; a generalization to n > 2 is obvious.

**Definition 10** **Numerical preference: P:= rank(F)(P1, P2)**

*Intuitively*: Not available in general.

*Formally:* We assume P1 := SCORE(A1, f1), P2 := SCORE(A2, f2) and a **combining function F:** $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$.

Let '<' denote the 'less-than' order on $\mathbb{R}$. For x = (x1, x2) and y = (y1, y2) $\in$ dom(A1) $\times$ dom(A2) we define: x <rank(F)(P1, P2) y iff F(f1(x1), f2(x2)) < F(f1(y1), f2(y2))

**P = (A1 $\cup$ A2, <rank(F)(P1, P2))** is called **numerical preference**.

The proof of strict partial order is immediate, since '<' is irreflexive and transitive. P need not be a chain, if F is not a one-to-one mapping. Also note that rank(F) is *not* an orthogonal preference constructor like '$\otimes$' or '&'. It can exclusively be applied to SCORE preferences. But vice versa, numerical preferences can be used as input to all other preference constructors.

**Example 5** **Numerical preference (F as weighted sum)**

Let's assume P1 := SCORE(A1: Integer, f1), f1(x) := distance(x, 0)

P2 := SCORE(A2: Integer, f2), f2(x) := distance(x, –2)

and P3 := rank(F)(P1, P2), F(x1, x2) := x1 + 2 $*$ x2

We study: 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 Ranking-R, containing for each value of R its score vector for f1, f2 together with its combined F-ranking:

Ranking-R = {(f12-val1 = (5, 5), F-val1 = 15), (f12-val2 = (5, 6), F-val2 = 17),
(f12-val3 = (5, 3), F-val3 = 11), (f12-val4 = (5, 8), F-val4 = 21),
(f12-val5 = (6, 2), F-val5 = 10), (f12-val6 = (6, 2), F-val6 = 10)}

The 'better-than' graph of P3 for subset R is not a chain and has 5 levels:

val4 $\to$ val2 $\to$ val1 $\to$ val3 $\to$ {val5, val6}

As an interesting observation the maximal f1-value being 6 does not show up in the top performer val4, having f12-val4 = (5,8). In some sense this is like discriminating against P1. ☺

### 3.3.2 Aggregating preference constructors

Aggregating preference constructors ($\blacklozenge$, +, $\oplus$) pursue a different, technical purpose. All proofs of partial order semantics are straightforward, following [DaP90]). Intersection '$\blacklozenge$' and disjoint union **'+'** perform a *"piece-wise" assembly* of 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* using '$\blacklozenge$' and '+'.

**Definition 11 Intersection and disjoint union preferences**

Let P1 = (A, <P1) and P2 = (A, <P2) be preferences on the same set of attribute names A.

a) **P = (A, < P1♦P2)** is called **intersection preference**, if: x <P1♦P2 y iff x <P1 y $\wedge$ x <P2 y

b) If P1 and P2 are *disjoint* preferences, then **P = (A, <P1+P2)** is called **disjoint union preference**, if:

x <P1+P2 y iff x <P1 y $\vee$ x <P2 y

**Definition 12 Linear sum preferences**

For single attributes A1 and A2 such that A1 ≠ A2 and dom(A1) ∩ dom(A2) = ∅ we assume P1 = (A1, <P1) and P2 = (A2, <P2), implying that P1 and P2 are disjoint preferences. For a new attribute name A we define dom(A) := dom(A1) ∪ dom(A2). Then **P = (A, <P1⊕P2)** is called **linear sum preference**, if:

x <P1⊕P2 y iff x <P1 y $\vee$ x <P2 y $\vee$ (x ∈ dom(A2) $\wedge$ y ∈ dom(A1))

Linear sum '⊕' can be viewed as a convenient design and proof method for base preference constructors. With the right understanding of 'other-values' (cmp. Definition 6) we can informally state:

- A POS-preference constructor can be characterized as the linear sum of the anti-chain preference on the POS-set followed by the anti-chain preference on the other values: POS = POS-set$^{\leftrightarrow}$ ⊕ other-values$^{\leftrightarrow}$
- POS/NEG = (POS-set$^{\leftrightarrow}$ ⊕ other-values$^{\leftrightarrow}$) ⊕ NEG-set$^{\leftrightarrow}$
- POS/POS = (POS1-set$^{\leftrightarrow}$ ⊕ POS2-set$^{\leftrightarrow}$) ⊕ other-values$^{\leftrightarrow}$
- EXPLICIT = E ⊕ other-values$^{\leftrightarrow}$

At this point we can summarize all results stated so far as follows, referring back to Definition 5:

**Proposition 1 Each preference term defines a preference.**

This proposition gives us the grand freedom to flexibly and intuitively 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 6 Preference engineering scenario**

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

P1 = (Category, <P1) := POS/POS(Category, POS-set1{cabriolet}; POS-set2{roadster})
P2 = (Transmission, <P2) := POS(Transmission, POS-SET{automatic})
P3 = (Horsepower, <P3) := AROUND(Horsepower, 100)
P4 = (Price, <P4) := LOWEST(Price)
P5 = (Color, <P6) := NEG(Color, NEG-set{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 this wish list to her car dealer Michael, who adds general *domain knowledge* P6 about cars:

P6 = (Year-of-construction, <P6) := HIGHEST(Year-of-construction)

In general, 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 = (Commission, <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 $\otimes$ P2 $\otimes$ P3) & P4)) & P6) & P7

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

P8 = (Color, <P8) := POS/NEG(Color, POS-set{blue}; NEG-set{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$\otimes$P8$\otimes$P4) & (P1$\otimes$P2$\otimes$P3)

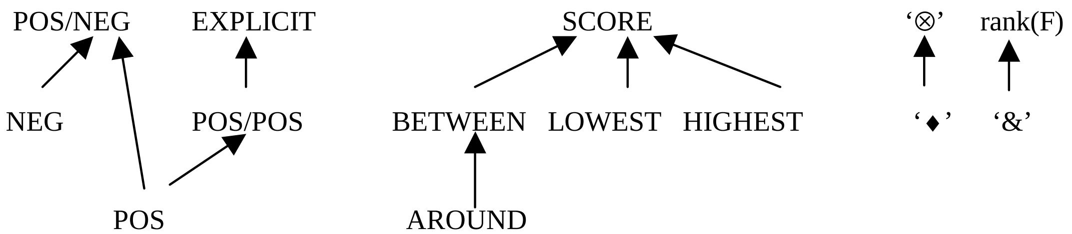
Finally Michael poses the adapted complex preference query $Q2^*$ … and the story might end that everybody is happy with the result. ☺

## 3.4 Preference hierarchies

Preference constructors can be arranged in hierarchies. Given constructors C1 and C2, we call C1 a **preference sub-constructor** of C2 (**C1 $\preccurlyeq$ C2**), if the definition of C1 can be gained from the definition of C2 by some specializing constraints. Basically we can state three hierarchies, where the latter will be proved later on.

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

These sub-constructor hierarchies can be visualized as follows:



There is certainly space for more sub-constructor relationships. For example, a super-constructor of both POS/NEG and EXPLICIT would be a constructor with two explicit graphs, say POS-graph and NEG-graph, assembled by linear sums in analogy to POS/NEG. An obvious possibility is to verify that '&' $\preccurlyeq$ rank(F) holds by determining a properly weighted F.

Since we have specialization by constraints, this sub-constructor hierarchy is taxonomic. Besides the usual advantages for 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 in database systems are basically formulated by first order logic formulas, which can be manipulated by Boolean algebra. On the other hand preferences, represented by preference terms, will be 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 13 Equivalence of preference terms**

P1 = (A1, <P1) and P2 = (A2, <P2) are **equivalent (P1 ≡ P2)**, if A1 = A2 and if for all x and y ∈ dom(A1):

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 by [DaP90].

**Proposition 2 Commutative and associative laws for preference terms**

| | | | |
|---|---|---|---|
| b) | Pareto accumulation: | P1 ⊗ P2 ≡ P2 ⊗ P1, | (P1 ⊗ P2) ⊗ P3 ≡ P1 ⊗ (P2 ⊗ P3) |
| c) | Prioritized accumulation: | | (P1 & P2) & P3 ≡ P1 & (P2 & P3) |
| d) | Intersection aggregation: | P1 ♦ P2 ≡ P2 ♦ P1, | (P1 ♦ P2) ♦ P3 ≡ P1 ♦ (P2 ♦ P3) |
| e) | Disjoint union aggregation: | P1 + P2 ≡ P2 + P1, | (P1 + P2) + P3 ≡ P1 + (P2 + P3) |
| f) | Linear sum aggregation: | | (P1 ⊕ P2) ⊕ P3 ≡ P1 ⊕ (P2 ⊕ P3) |

For numerical accumulation the existence of such algebraic laws depends on the mathematical properties of F.

**Proposition 3** **Further laws for preference terms**

Dual preferences: a) $(S^{\leftrightarrow})^{\partial} \equiv S^{\leftrightarrow}$ for any set S b) $(P^{\partial})^{\partial} \equiv P$

c) $(P1 \oplus P2)^{\partial} \equiv P2^{\partial} \oplus P1^{\partial}$ d) HIGHEST $\equiv$ LOWEST$^{\partial}$

e) $POS^{\partial} \equiv NEG$, $NEG^{\partial} \equiv POS$ if POS-set = NEG-set

Intersection aggregation: f) $P \blacklozenge P \equiv P$ g) $P \blacklozenge P^{\delta} \equiv P \blacklozenge A^{\leftrightarrow} \equiv A^{\leftrightarrow}$ if $P = (A, <P)$

Prioritized accumulation: h) If P1 and P2 are chains, then P1&P2 and P2&P1 are chains

i) $P \,\&\, P \equiv P \,\&\, P^{\partial} \equiv P$ j) $P \,\&\, A^{\leftrightarrow} \equiv P$ if $P = (A, <P)$

k) $A^{\leftrightarrow}\&P \equiv A^{\leftrightarrow}$ if $P = (A, <P)$

Pareto accumulation: l) $P \otimes P \equiv P$ m) $A^{\leftrightarrow}\otimes P \equiv A^{\leftrightarrow}\&P$

n) $P \otimes A^{\leftrightarrow} \equiv P \otimes P^{\partial} \equiv A^{\leftrightarrow}$ if $P = (A, <P)$

These laws are easy to obtain and they all match our intuitive expectations about the semantics of preferences. E.g., let's pick $P \otimes P^{\partial} \equiv A^{\leftrightarrow}$: Since P and $P^{\partial}$ are equally important, in case of conflicts for values x and y none of them prevails, instead x and y remain unranked. Because P and $P^{\partial}$ are in conflict everywhere, the full domain becomes unranked, hence the anti-chain $A^{\leftrightarrow}$. Unranked values are a natural reservoir to negotiate compromises.

## 4.2 Decomposition of prioritized and Pareto preferences

The following "discrimination" theorem corresponds to the intuitive semantics of prioritized accumulation. We succeed to decompose '&' into disjoint union aggregation.

**Proposition 4** **"Discrimination" theorem for P1&P2:**

**(a)** $\mathbf{P1\&P2 \equiv P1}$ **if** $\mathbf{P1 = (A, <P1)}$ **and** $\mathbf{P2 = (A, <P2)}$

**(b)** $\mathbf{P1\&P2 \equiv P1 + (A1^{\leftrightarrow}\&P2)}$ **if** $\mathbf{A1 \cap A2 = \varnothing}$

Proof: See appendix.

In both cases P1 is fully respected. For shared attributes P2 is completely dominated by P1. In case of disjoint attributes 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.

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

**Proposition 5** **"Non-discrimination" theorem:** $\mathbf{P1 \otimes P2 \equiv (P1 \,\&\, P2) \blacklozenge (P2 \,\&\, P1)}$

Proof: See appendix.

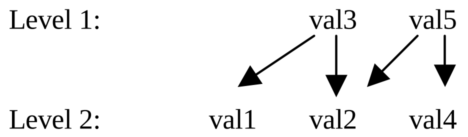
Preferences P1 and P2 are indeed treated *equally important* by '⊗', since both P1 and P2 are each given prime importance by '&'. Any arising conflict is resolved in a non-discriminating way by intersection '♦'.

**Example 7 "Non-discrimination" theorem**

Let's assume P1 = LOWEST(Price), P2 = LOWEST(Mileage) and a Pareto preference P = ({Price, Mileage}, <P1⊗P2). We consider this set Car-DB of values 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 (gained 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 as follows:

val5 $\rightarrow$ val4 $\rightarrow$ val3 $\rightarrow$ val2 $\rightarrow$ val1

The 'better-than graph' of P'' = P2&P1 for subset Car-DB yields a chain as follows:

val3 $\rightarrow$ val1 $\rightarrow$ val5 $\rightarrow$ val2 $\rightarrow$ 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 that are shared by P' and P''. ☺

**Proposition 6 P1⊗P2 ≡ P1♦P2 if P1 = (A, <P1) and P2 = (A, <P2)**

Proof: : Direct corollary from Proposition 5 and Proposition 4 a). Thus ♦ is a preference sub-constructor of ⊗.

# 5 Evaluation of preference queries

If we look at SQL databases, then life is comparably simple there. Queries against a database set 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 were asking for, otherwise you get nothing at all. We call the latter deficiency the *empty-result problem*. Thus 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 means of 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 more intelligent and *cooperative answer semantics* of preference queries 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 *matchmaking* between wishes and reality. To this purpose we now define the so-called *BMO query model*.

## 5.1 Preference queries and the BMO query model

Preferences are defined in terms of values from dom(A), which represent the universe of a fictitious world (realm of wishes). In database applications we assume that the real world is mapped into appropriate database instances which we call database sets. The database set R may, e.g., be a view or a base relation in an SQL database or a DTD-instance in an XML database. Under the usual closed world assumption database sets capture the currently valid or accessible state of the real world. Thus database sets are proper 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 $\pi$ of R onto these k attributes.

**Definition 14 Database preference $P^R$, perfect match for P in 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 $P^{\subseteq}$. We call it a **database preference** and denote it 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 15 Declarative semantics of a preference query σ[P](R), BMO query model**

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

A preference query 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 8 BMO query model**

We revisit the EXPLICIT preference P of Example 1 and pose the query σ[P](R) for R(Color) = {yellow, red, green, black}. The BMO result is: σ[P](R) = {yellow, red}. Note that red is a perfect match. ☺

The next proposition is straightforward, but important to state.

**Proposition 7 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. To this purpose consider the preference query $\sigma[A^{\leftrightarrow}\&P](R)$, where $P = (B, <P)$:

We have : $x <A^{\leftrightarrow}\&P\ y$ iff $x1 <A^{\leftrightarrow} y1 \vee (x1 = y1 \wedge x2 <P\ y2)$

iff $false \vee (x1 = y1 \wedge x2 <P\ y2)$ iff $x1 = y1 \wedge x2 <P\ y2$

Then: $t \in \sigma[A^{\leftrightarrow}\&P](R)$ iff $t[A, B] \in \max((A^{\leftrightarrow}\&P)^R)$

iff $\forall v[A, B] \in R[A, B]: \neg(t[A, B] <_{A^{\leftrightarrow}\&P} v[A, B])$

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 16 Declarative semantics of σ[P groupby A](R)**

Let's assume $P = (B, <_P)$ and a database preference $P^R$. We declaratively define a **preference query with grouping σ[P groupby A](R)** as follows: $\sigma[P \text{ groupby } A](R) := \sigma[A^{\leftrightarrow}\&P](R)$

Compared to hard selection queries, preference selection queries deviate from the logics behind hard selections: Preference queries are always **non-monotonic**.

**Example 9 Non-monotonicity of preference query results**

Let's consider P = HIGHEST(Fuel_Economy) ⊗ HIGHEST(Insurance_Rating). We successively evaluate σ[P](Cars) for Cars(Fuel_Economy, Insurance_Rating, Nickname) as follows:

Cars = {(100, 3, frog), (50, 3, cat)}: σ[P](R) = {(100, 3, frog)}

Cars = {(100, 3, frog), (50, 3, cat) (50, 10, shark)}: σ[P](R) = {(100, 3, frog), (50, 10, shark)}

Cars = {(100, 3, frog), (50, 3, cat) (50, 10, shark), (100, 10, turtle)}: σ[P](R) = {(100, 10, turtle)} ☺

The non-monotonic behavior is obvious: Though we added more and more tuples to Cars, the results of our preference queries did not exhibit a similar behavior. Instead of adapting to the size of the database set Cars, query results of σ[P](R) *adapted to the quality* of data in Cars.

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 queries. Or considered from a different perspective, *better data imply better query results.* As it is always the case in the real world, the law of energy preservation applies here, too: Evaluation of preference queries is potentially more expensive than of hard selection queries, because non-monotonic logics leads to more complex evaluation algorithms in general.

Thus one key challenge of preference query evaluation is to find efficient algorithms for complex preference constructors. For a Pareto preference, e.g, the naïve approach performs $O(n^2)$ 'better-than' tests, if the database set R has n tuples. For the scope of this paper, however, we do not explicitly address efficiency issues, instead we provide fundamental decomposition results that can form the basis for a divide-and-conquer approach pursued by a preference query optimizer.

## 5.2 Evaluation of disjoint union and intersection aggregation

Our goal is to decompose the accumulation preference constructors '&' and '⊗' into aggregation accumulation using '+' and '♦', which in turn can be decomposed further.

**Proposition 8** $\sigma[P1+P2](R) = \sigma[P1](R) \cap \sigma[P2](R)$

Proof: See appendix.

The evaluation of intersection aggregation will cause more headaches. First we need some technical definitions.

**Definition 17** $\mathbf{Nmax(P^R),\ {}_P{\uparrow}v,\ \ YY(P1, P2)^R}$

a) Given P = (A, <P) and a database preference $P^R$, the set of **non-maximal values** $Nmax(P^R)$ is defined as:

$$Nmax(P^R) := R[A] - max(P^R)$$

b) Given $v \in dom(A)$, the following set is called **'better than' set** of v in P: ${}_P{\uparrow}v := \{w \in dom(A): v <_P w\}$

c) $YY(P1, P2)^R := \{t \in R : t[A] \in Nmax(P1^R) \cap Nmax(P2^R) \ \wedge\ {}_{P1}{\uparrow}t[A] \ \cap\ {}_{P2}{\uparrow}t[A] = \varnothing\}$

In the terminology of [DaP90] $Nmax(P^R)$ is a *down-set* (or *order ideal*), whereas $max(P^R)$ is an *up-set* (or *order filter*). Likewise, ${}_P{\uparrow}v$ is an up-set.

**Proposition 9** $\sigma[P1\blacklozenge P2](R) = \sigma[P1](R) \cup \sigma[P2](R) \cup YY(P1, P2)^R$

Proof: See appendix.

Efficiently evaluating $YY(P1, P2)^R$ is a difficult recursive task in general. Therefore it is an interesting future challenge to figure out conditions under which $YY(P1, P2)^R$ can be simplified.

## 5.3 Evaluation of prioritized accumulation

Next we investigate σ[P1&P2](R). Since P1&P2 ≡ P1 for shared attributes (Proposition 4 a) we assume $A1 \cap A2 = \varnothing$. The evaluation of prioritized accumulation can be done by **grouping.**

**Proposition 10** $\sigma[P1\&P2](R) = \sigma[P1](R) \cap \sigma[P2 \text{ groupby } A1](R)$**, if** $A1 \cap A2 = \varnothing$

Proof: Let P1 = (A1, <P1) and P2 = (A2, <P2). From Proposition 4 b, Proposition 8 and Definition 16 we get:

$$\sigma[P1\&P2](R) = \sigma[P1+(A1^{\leftrightarrow}\&P2)](R) = \sigma[P1](R) \cap \sigma[A1^{\leftrightarrow}\&P2)](R)$$

$$= \sigma[P1](R) \cap \sigma[P2 \text{ groupby } A1](R) \qquad \text{Q.e.d.}$$

**Example 10** **Evaluation of a prioritized accumulation query**

We assume $P1 = Make^{\leftrightarrow}$, P2 = AROUND(Price, 40000) and this 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 an offer with a price around 40000" translates into:

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

$$= Cars \cap \{(Audi, 40000, 1), (BMW, 35000, 2), (VW, 20000, 3)\}$$

$$= \{(Audi, 40000, 1), (BMW, 35000, 2), (VW, 20000, 3)\}$$ ☺

**Proposition 11** $\sigma[P1\&P2](R) = \sigma[P2](\sigma[P1](R))$**, if P1 is a chain**

Proof: If P1 is a chain, all tuples in σ[P1](R) have the same A1-value. Then Proposition 10 specializes as stated. Thus a **cascade** of preference queries is a special case of a prioritized preference query, if P1 is a chain.

## 5.4 Evaluation of Pareto accumulation

Now we state the main decomposition theorem for the evaluation of Pareto preference queries.

**Proposition 12** $\boldsymbol{\sigma[P1\otimes P2](R) = (\sigma[P1](R) \cap \sigma[P2 \text{ groupby } A1](R)) \cup}$

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

Proof: Due to Proposition 5, Proposition 9 and Proposition 10 we get:

$$\begin{aligned}\sigma[P1\otimes P2](R) &= \sigma[(P1\&P2) \blacklozenge (P2\&P1)](R) \\ &= \sigma[P1\&P2](R) \cup \sigma[P2\&P1](R) \cup YY(P1\&P2, P2\&P1)^R \\ &= (\sigma[P1](R) \cap \sigma[P2 \text{ groupby } A1](R)) \cup \\ &\quad (\sigma[P2](R) \cap \sigma[P1 \text{ groupby } A2](R)) \cup YY(P1\&P2, P2\&P1)^R \qquad \text{Q.e.d.}\end{aligned}$$

This theorem gives a good insight into the structure of the Pareto-optimal set $\sigma[P1\otimes P2](R)$, re-enforcing also our claim that '$\otimes$' treats P1 and P2 as equally important:

- The first term contains all maximal values of $(P1\&P2)^R$.
- The 2nd term contains all maximal values of $(P2\&P1)^R$.
- 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 Proposition 11 can be applied to speed up evaluation.

**Example 11** **Evaluation of Pareto accumulation**

Assume P1 = LOWEST(A), the dual preference P2 = HIGHEST(A) and R(A) = {3, 6, 9}. We compute $\sigma[P1\otimes P2](R)$. Due to Proposition 6, Proposition 3d, g) we immediately know:

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

To countercheck, since both P1 and P2 are chains Proposition 12 specializes as follows:

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

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

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

Thus we finally arrive at: $\sigma[P1\otimes P2](R) = \{3\} \cup \{9\} \cup \{6\} = R$ ☺

## 5.5 Filter effect of Pareto accumulation

Preference queries under the BMO query model avoid both the empty-result effect and the flooding effect with irrelevant results. We want to study how the filter effect of a Pareto preference can be characterized.

**Definition 18** **Result size**

Let P = (A, <P). The **result size** of $\sigma[P](R)$ is defined as: $\text{size}(P, R) := \text{card}(\pi_A(\sigma[P](R)) = \text{card}(\max(P^R))$

Size(P, R) counts the number of different A-values appearing in the result of a preference query under the BMO query model. Obviously if card(R) > 0, then $1 \le \text{size}(P, R) \le \text{card}(\pi_A(R))$.

**Definition 19 Strength of a preference filter**

Given P1 = (A, <P1) and P2 = (A, <P2), we say that P1 is a **stronger preference filter** than P2, if size(P1, R) ≤ size(P2, R). Conversely, P2 is said to be a **weaker preference filter** than P1. Note that 'stronger than' is a (non-strict) partial order on the set of all preferences, given A and R.

**Proposition 13 Result sizes of complex preferences**

a) size(P1+P2, R) ≤ size(P1, R), size(P1+P2, R) ≤ size(P2, R)
b) size(P1♦P2, R) ≥ size(P1, R), size(P1♦P2, R) ≥ size(P2, R)
c) size(P1&P2, R) ≤ size(P1, R)
d) size(P1⊗P2, R) ≥ size(P1&P2, R), size(P1⊗P2, R) ≥ size(P2&P1, R)

Proof:

a) Let P1 = (A, <P1) and P2 = (A, <P2), where P1and P2 are disjoint preferences. From Proposition 8 we get:

$$\text{size}(P1+P2, R) = \text{card}(\pi_A(\sigma[P1+P2](R))) = \text{card}(\pi_A(\sigma[P1](R) \cap \sigma[P2](R))) \le \text{card}(\pi_A(\sigma[P1](R))) = \text{size}(P1, R)$$

b) Let P1 = (A, <P1) and P2 = (A, <P2). Then due to Proposition 9 we get:

$$\text{size}(P1\blacklozenge P2, R) = \text{card}(\pi_A(\sigma[P1\blacklozenge P2](R))) = \text{card}(\pi_A(\sigma[P1](R) \cup \sigma[P2](R) \cup YY(P1, P2)^R)) \ge \text{card}(\pi_A(\sigma[P1](R))) = \text{size}(P1, R)$$

c) Let P1 = (A1, <P1), P2 = (A2, <P2) and A = A1 ∪ A2. Then due to Proposition 10 we get:

$$\text{size}(P1\&P2, R) = \text{card}(\pi_A(\sigma[P1\&P2](R))) = \text{card}(\pi_A(\sigma[P1](R) \cap \sigma[P2 \text{ groupby } A1](R))) \le \text{card}(\pi_A(\sigma[P1](R))) = \text{size}(P1, R)$$

d) Let P1 = (A1, <P1) and P1 = (A2, <P1). Then due to Proposition 5 and Proposition 13 b) we get:

$$\text{size}(P1\otimes P2, R) = \text{size}((P1\&P2)\blacklozenge(P2\&P1), R) \ge \text{size}(P1\&P2, R)$$

Since '+', '♦' and '⊗' are commutative, the remaining inequalities holds, too. Q.e.d.

Using the notation "P ⇛ Q iff P is a stronger preference filter than Q, given A and R", we thus can state:

P1+P2 ⇛ P1, P1+P2 ⇛ P2, P1 ⇛ P1♦P2, P2 ⇛ P1♦P2, P1&P2 ⇛ P1,

P1&P2 ⇛ P1⊗P2, P2&P1 ⇛ P1⊗P2

We want to 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 have:

P1⊗P2 ⇚ P1&P2 ⇛ P1, P1⊗P2 ⇚ P2&P1 ⇛ P2

This behavior justifies the following interpretation: Seen from the perspective of P1 and P2, resp., forming P1&P2 and P2&P1 has a stronger filter effect, hence resembling 'AND' operations in the exact match query model. Then continuing to form P1⊗P2 has a weaker filter effect, hence resembling 'OR' operations in the exact match query model. Since the BMO query model automatically adapts to the quality of the database set R, as a net effect we get an **automatic 'AND/OR'-like filter effect of Pareto accumulation**.

This is an important observation compared to search engines with an exact match query model, struggling to combat the empty-result nuisance and the flooding effect. There are work-arounds that attempt to mitigate this, e.g. *parametric search*, which basically is a semi-automatic, repetitive attempt of query refinement. The other countermeasure is the so-called '*expert mode'*, offering a Boolean query interface with logical AND, OR and NOT operations. However, this approach has been known as inadequate for a long time ([VGB61]). BMO databases take all this burden from the user by *automatically* and *adaptively* finding the best possible answers.

# 6 Practical aspects

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

## 6.1 Integration into SQL and XML

The early origins that eventually led to the contributions of this paper trace back to [KiG94], proposing a deductive approach to programming with preferences as partial orders. The theory of subsumption lattices in [KKT95] can provide the formal backbone guaranteeing the existence of a model theory and a corresponding fixpoint theory with subsumption in general deductive databases. Subsumption lattices generalize the usual powerset lattices of Herbrand models to arbitrary partial orders. In turn, Herbrand models, representing the exact match query model, are a special case of subsumption models. As a consequence this meets one *crucial point* from our introductory list of desiderata: We can *compatibly* extend declarative database query languages under an exact query model, which includes object-relational SQL databases and XML databases, by preferences under a BMO query model.

- **Preference SQL**

The product Preference SQL, whose first release was available already in the fall of 1999, has been the first instance of an extension of SQL by preferences as strict partial orders. No publications have been released in the past, but recently [KiK01] gives an overview. Preference SQL implements a plug-and-go application integration by a clever rewriting of Preference SQL queries into SQL92 code, making it available e.g. on DB2, Oracle 8i and the MS SQL Server. 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') and cascading preferences. Preferences can be applied following a new PREFERRING clause. Here are two self-explaining examples:

```
SELECT * FROM car WHERE make = ‘Opel’
PREFERRING(category = 'roadster' ELSE category <> 'passenger' AND
           price AROUND 40000 AND HIGHEST(power))
           CASCADE color = 'red' CASCADE LOWEST(mileage);
SELECT * FROM trips
PREFERRING start_date AROUND '2001/11/23' AND duration AROUND 14
BUT ONLY   DISTANCE(start_date)<=2 AND DISTANCE(duration)<=2;
```

The quality functions `LEVEL` and `DISTANCE` can supervise required quality levels (`BUT ONLY` clause) and can be exploited for advanced query explanation. In [KFH01], describing experiences from a smart meta-comparison shop using Preference SQL, benchmarks from real customer queries show 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.

- **Preference XPATH**

Preference XPATH ([KHF01]) is a query language to build personalized query engines in an attribute-rich XML environment. It implements the full presented preference model (currently up rank(F)), the prototype runs on the native XML database system TAMINO from Software AG and on XALAN. Standard XPATH is compatibly 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 the symbols '[' and ']'. For soft selections (i.e. `preference`) '#[' and ']#' are used. Here are two sample queries, where Pareto accumulation is written as 'AND' and prioritized accumulation is expressed by '`PRIOR TO`':

```
Q1: /CARS/CAR #[(@fuel_economy)highest and (@horsepower)highest]#
Q2: /CARS/CAR #[(@color)in("black", "white")prior to(@price)around 10000]#
              #[(@mileage)lowest]#
```

Preference XPATH can be applied in other XML key technologies like XSLT, Xpointer or Xquery.

- **The 'skyline of' clause**

A restricted form of Pareto accumulation is the 'SKYLINE OF' clause proposed in [BKS01]. It is a non-strict variant for specifying $P = P_1 \otimes P_2 \otimes \ldots \otimes P_k$, where each $P_i$ must be a LOWEST or HIGHEST preference, hence a chain. Efficient evaluation algorithms have been given in [KLP75], [BKS01] and [TEO01].

## 6.2 The ranked query model

Soft constraints implemented by numerical accumulation rank(F) are in use today in several database and information retrieval applications.

- **Multi-feature query engines**

One typical use is in *multi-feature query engines* to rank objects according to a complex technical features, e.g. to support queries by image content on color, texture or shape. Since rank(F) constructs chain preferences in most cases, a BMO-query semantics would return exactly one best-matching object. Definitely, this is a too small set to choose from in general. For more alternative choices, the "*k-best*" query model is applied, returning k objects with a (user-)definable k. In BMO-terms this amounts to retrieve some non-maximal objects, too. There is already the SQL/MM proposal for incorporating multi-feature queries into SQL. Algorithms like Quick-Combine ([GBK00, BGK00]) can be used to speed up the computation of rank(F) under the "k-best" semantics.

- **Full-text search engines**

Another area are *full-text search engines*, where search keywords can be understood as special preferences, each yielding a numerical score indicating their *relevance*. The combining function F for rank(F) is typically some monotonic scalar product employing the cosine function, if the classical vector space model from information retrieval is used. SQL has been extended by a text cartridge (Oracle 8i), a text extender (DB2) or text datablade (Informix), implementing a k-best query model. The XXL prototype of [ThW00] is a representative of providing the k-best semantics in the XML context.

Let's spend a word on the issue of non-numerical vs. numerical ranking or of attribute-based search vs. full-text search: If efficient implementations of the full preference model are available, then there are much more options how to model preferences in a given application, ranging from purely non-numerical to purely numerical and any combinations in between. For instance, an interesting combination of attribute-rich search and full-text search would be a synthesis of Preference XPATH and XXL. Likewise Preference SQL merges well with SQL and ranked text (cartridges, extenders, datablades). Ranked text itself may be one feature in a multi-feature query. Theoretically it may be the case that numerical ranking subsumes all other preferences (which amounts to prove that every preference constructor is a sub-constructor of SCORE or rank(F)). However, it is preferable to support a *plurality of preference constructors*: Identify as many preference constructors as possible that (1) frequently occur in the real world, (2) have an intuitive semantics and (3) possess efficient evaluation algorithms.

# 7 Summary and outlook

We presented a preference model which is tailored for database systems. Many requirements of a personalized real world are met by preferences modeled as strict partial orders: It unifies non-numerical and numerical ranking, it has an intuitive semantics that is understood by everybody and it can be mapped directly into a well-developed mathematical framework. This preference model features a variety of preference constructors that are frequently needed in practice. In particular we introduced Pareto accumulation for equally important preferences and prioritized accumulation for ordered importance among preferences. Intended for technical assembly we showed that the preference constructors of disjoint union, linear sum and intersection aggregation fit into this framework, too. This wide spectrum of options to model preferences opens the door for a systematic approach to preference engineering, where preferences can be combined inductively, including situations where they come from different parties with potentially conflicting intentions. Such a preference mix may be comprised of subjective preferences from daily life experiences, driven by personal intentions, and of sophisticated technical preferences. Various portions of the presented preference model have already been prototyped or are in commercial use in SQL or XML environments. Despite this vast application scope, preferences as strict partial orders possess an almost Spartan formal basis. This simplicity in turn is the key for a preference algebra, where many laws are valid that are of interest for a preference optimizer. We have given a collection of laws which all have an intuitive interpretation, a key proposition being the non-discrimination theorem for Pareto accumulation. This formal basis enabled us to define the declarative semantics of preference queries under the BMO query model, which can cope with the notorious empty-result and flooding problems of search engine technology. Moreover, we succeeded to present fundamental decomposition theorems for the evaluation of Pareto and of prioritized preference queries. Beyond the scope of this paper, however, has been the issue of efficiency of preference query evaluation. Due to the inherent non-monotonic nature of preference queries, this is a major challenge.

Our roadmap into a "*Preference World*" includes the following investigations: A persistent preference repository, personalized query composition methods, preference mining from query log files, a preference query optimizer (e.g. heuristic transformations like 'push preference', cost-based optimization to choose between direct implementations of the Pareto operator and divide & conquer algorithms exploiting the decomposition principles, or the use of index methods for efficient 'better-than' testing). The conflict tolerance of our preference model forms the basis for research concerned with e-negotiations and e-haggling. Finally, we work on enhancements of our preference model to incorporate additional intuitive semantic features. Eventually in a Preference World technologies comparable to the exact-match world should become available, ranging from E/R/Preference modeling to efficient and scalable preference query languages for SQL and XML.

**Acknowledgments:**

I would like to thank Thorsten Ehm and Bernhard Möller for carefully reading a draft of this paper, giving valuable suggestions to improve its technical presentation. Likewise I appreciate the cooperation of Ulrich Güntzer in the Preference World project and the contributions of Gerhard Köstler throughout the Preference SQL project.

## Literature:

[ACM00] Special issue of the Communications of the ACM on Personalization, vol. 43, Aug. 2000.

[Arr59] K. Arrow: *Rational Choice Functions and Orderings.* Economica 26: pp. 121–127, 1959.

[BGK00] W-T. Balke, U. Güntzer, W. Kießling: *Applications of Quick-Combine for Ranked Query Models*. Proc. 1st DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries, Zurich, 2000.

[BKS01] S. Borzsonyi, D. Kossmann, K. Stocker: *The Skyline Operator.* Proc. 17th Intern. Conf. On Data Engineering, Heidelberg, Germany, April 2001.

[BLL01] M. Bichler, J. Lee, H.S. Lee, J.-Y. Chung: *ABSolute: An Intelligent Decision Making Framework for E-Sourcing.* Proc. 3rd Intern. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, pp. 21-30, San Jose, USA, June 2001.

[CYC96] W. W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, C. Larson: *CoBase - A Scalable and Extensible Cooperative Information System*. Journal of Intelligent Information Systems, 6(3):223-259, 1996.

[DaP90] B.A. Davey, H.A. Priestley: *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge University press, 1990.

[GaL94] T. Gaasterland, J. Lobo: *Qualified Answers that Reflect User Needs and Preferences*. Proc. VLDB 1994, Santiago de Chile.

[GBK00] U. Güntzer, W.-T. Balke, W. Kießling: *Optimizing Multi-Feature Queries for Image Databases*. Proc. 26th Intern. Conf. on Very Large Databases (VLDB 2000), pages 419-428, Cairo, Egypt, 2000.

[KeR93] R. Keeney, H. Raiffa: *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press, UK, 1993.

[KiG94] 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, Springer LNCS 777, pp. 118-138, Hamburg, 1994.

[KiK01] W. Kießling, G. Köstler: *Preference SQL – Design, Implementation, Experiences*. Technical report 2001-7, Institute of Computer Science, Univ. of Augsburg, Oct. 2001, submitted for publication.

[KiQ01] A. Kiss, J. Quinqueton: *Multiagent Cooperative Learning of User Preferences*, 5th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Freiburg, Sept. 2001.

[KFH01] 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 Information Systems, pp. 21-30, San Jose, USA, June 2001.

[KHF01] W. Kießling, B. Hafenrichter, S. Fischer, S. Holland: *Preference XPATH: A Query Language for E-Commerce*. Proc. 5th Intern. Konferenz für Wirtschaftsinformatik, Augsburg, Germany, pp. 425-440, Sept. 2001.

[KKT95] 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.

[KLP75] H. T. Kung, F. Luccio, F.P. Preparata: *On Finding the Maxima of a Set of Vectors.* Journal of the ACM, 22(4): 469-476, 1975.

[Min98] 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.

[Mot88] A. Motro: *VAGUE: A User Interface to Relational Databases that Permits Vague Queries.* ACM Transactions on Office Automation Systems, Vol. 6, No. 3, pp. 187-214, 1988.

[PeS01] J. E. Peris, C. Sanchez: *Characterization of Social Choice Sets in terms of Individual's Maximal Sets: The Fixed Agenda Framework*. Springer, Social Choice and Welfare (2001) 18: pp. 113 – 127.

[TEO01] K.-L. Tan, P.-K. Eng, B. C. Ooi: *Efficient Progressive Skyline Computation*. Proc. 27th Intern. Conf. on Very Large Databases, pp. 301-310, Rome, Italy, Sept. 2001.

[ThW00] A. Theobald, G. Weikum: *Adding Relevance to XML.* Proc. of the 3rd Intern. Workshop on the Web and Databases, LNCS, Springer, 2000.

[VGB61] J. Verhoeff, W. Goffmann, J. Belzer: *Inefficiency of the Use of Boolean Functions for Information Retrieval Systems*, Comm. of the ACM, Dec. 1961, Vol. 4, No. 2.

## Appendix:

**(A) "Discrimination" theorem for P1&P2:**

**(a) $P1\&P2 \equiv P1$ if $P1 = (A, <_{P1})$ and $P2 = (A, <_{P2})$**

**(b) $P1\&P2 \equiv P1 + (A1^{\leftrightarrow}\&P2)$ if $A1 \cap A2 = \varnothing$**

Proof:

(a) Let $P1 = (A, <_{P1})$ and $P2 = (A, <_{P2})$. Then $P1\&P2 = (A, <_{P1\&P2})$. For $x, y \in dom(A)$ we get:

$$x <_{P1\&P2} y \text{ iff } x <_{P1} y \lor (x = y \land x <_{P2} y) \text{ iff } x <_{P1} y \lor \text{false} \text{ iff } x <_{P1} y$$

(b) Let $P1 = (A1, <_{P1})$ and $P2 = (A2, <_{P2})$ where $A1 \cap A2 = \varnothing$. For $x = (x1, x2)$, $y = (y1, y2) \in dom(A1) \times dom(A2)$ let $x <_{P1^*} y$ iff $x1 <_{P1} y1$. Then P1 is an order embedding into $P1^* = (A1 \cup A2, <_{P1^*})$.
Since $A1 \cap A2 = \varnothing$, $P1^*$ and P2 are disjoint preferences, hence $P1^*$ and $A1^{\leftrightarrow}\&P2$ are disjoint, too.

Thus $P1^* + (A1^{\leftrightarrow} \& P2) = (A1 \cup A2, <P1^*+(A1^{\leftrightarrow}\&P2))$ is a disjoint union preference. Now we get:

$$x <_{P1+(A1^{\leftrightarrow}\&P2)} y \text{ iff } x <_{P1^*+(A1^{\leftrightarrow}\&P2)} y \text{ iff } x1 <_{P1} y1 \vee (x <_{A1^{\leftrightarrow}\&P2} y)$$

$$\text{iff } x1 <_{P1} y1 \vee (x1 = y1 \wedge x2 <_{P2} y2) \text{ iff } x <_{P1\&P2} y \qquad \text{Q.e.d.}$$

**(B) "Non-discrimination" theorem: $P1 \otimes P2 \equiv (P1 \& P2) \blacklozenge (P2 \& P1)$**

Proof:

Let P1 = (A1, <P1) and P2 = (A2, <P2). Then:

$P1 \otimes P2 = (A1 \cup A2, <P1\otimes P2)$

$P1 \& P2 = (A1 \cup A2, <P1\&P2)$, $P2 \& P1 = (A2 \cup A1, <P2\&P1)$

$(P1 \& P2) \blacklozenge (P2 \& P1) = (A1 \cup A2, <(P1\&P2)\blacklozenge(P2\&P1))$

Let $x = (x1, x2)$ and $y = (y1, y2) \in \text{dom}(A1) \times \text{dom}(A2)$. For abbreviation let:

B := 'x1 <P1 y1', C := 'x1 = y1', D := 'x2 <P2 y2', E := 'x2 = y2'

If x = y, then $\neg B \wedge \neg D$ holds. On the other hand, if $x \neq y$, then $\neg C \vee \neg D$ holds. (*)

(1) $x <_{P1\otimes P2} y$ iff $(B \wedge (E \vee D)) \vee (D \wedge (C \vee B))$

iff $((B \wedge E) \vee (B \wedge D)) \vee ((D \wedge C) \vee (D \wedge B))$

iff $(B \wedge E) \vee (B \wedge D) \vee (D \wedge C)$

(2) $x <_{(P1\&P2)\blacklozenge(P2\&P1)} y$

iff $(B \vee (C \wedge D)) \wedge (D \vee (E \wedge B))$

iff $(B \wedge (D \vee (E \wedge B))) \vee ((C \wedge D) \wedge (D \vee (E \wedge B)))$

iff $(B \wedge D) \vee (B \wedge E \wedge B) \vee (C \wedge D \wedge D) \vee ((C \wedge E) \wedge D \wedge B)$

iff $x <_{P1\otimes P2} y \vee ((C \wedge E) \wedge D \wedge B)$ (**)

Now take a closer look at the last disjunctive term $H := C \wedge E \wedge D \wedge B$ in (**) above: In both cases that $x \neq y$ or $x = y$, due to (*) $\neg H$ holds. Therefore immediately from Boolean algebra we can continue (**):

iff $x <_{P1\otimes P2} y$ Q.e.d.

**(C) Theorem: $\sigma[P1+P2](R) = \sigma[P1](R) \cap \sigma[P2](R)$**

Proof: Consider $P1+P2 = (A, <P1+P2)$, the database preference $(P1+P2)^R$ and $w \in R[A]$:

$w \in \text{Nmax}((P1+P2)^R)$ iff $\exists v \in R[A]$: $w <_{P1+P2} v$ iff $\exists v \in R[A]$: $w <_{P1} v \vee w <_{P2} v$

Since P1 and P2 have to be disjoint preferences, we can continue:

iff $(\exists v \in R[A]: w <_{P1} v) \vee (\exists v \in R[A]: w <_{P2} v)$

iff $w \in \text{Nmax}(P1^R) \vee w \in \text{Nmax}(P2^R)$

Thus: $\text{Nmax}((P1+P2)^R) = \text{Nmax}(P1^R) \cup \text{Nmax}(P2^R)$

Then: $\sigma[P1+P2](R) = \{t \in R: t[A] \in \max((P1+P2)^R)\}$

$= \{t \in R: t[A] \in R[A] - \text{Nmax}((P1+P2)^R)\}$

$= \{t \in R: t[A] \in R[A] - (\text{Nmax}(P1^R) \cup \text{Nmax}(P2^R))\}$

$= \{t \in R: t[A] \in (R[A] - Nmax(P1^R)) \cap (R[A] - Nmax(P2^R))\}$

$= \{t \in R: t[A] \in max(P1^R) \cap max(P2^R)\} = \sigma[P1](R) \cap \sigma[P2](R)$ Q.e.d.

**(D) Theorem:** $\boldsymbol{\sigma[P1 \blacklozenge P2](R) = \sigma[P1](R) \cup \sigma[P2](R) \cup YY(P1, P2)^R}$

Proof: Consider $P1 \blacklozenge P2 = (A, <P1 \blacklozenge P2)$, the database preference $(P1 \blacklozenge P2)^R$ and $w \in R[A]$:

$w \in Nmax((P1 \blacklozenge P2)^R)$ iff $\exists v \in R[A]$: $w <P1 \blacklozenge P2\ v$ iff $\exists v \in R[A]$: $w <P1\ v \wedge w <P2\ v$

At this point we must be careful when distributing the existential quantifier into the conjunction:

iff $\exists v, v' \in R[A]$: $w <P1\ v \wedge w <P2\ v' \wedge (v \in {}_{P1}{\uparrow}w \wedge v' \in {}_{P2}{\uparrow}w \wedge v = v')$

iff $(\exists v \in R[A]: w <P1\ v) \wedge (\exists v' \in R[A]: w <P2\ v') \wedge$

$(\exists v \in Nmax(P1^R), \exists v' \in Nmax(P2^R): v \in {}_{P1}{\uparrow}w \wedge v' \in {}_{P2}{\uparrow}w \wedge v = v')$

iff $w \in Nmax(P1^R) \wedge w \in Nmax(P2^R) \wedge$

$(\exists v \in Nmax(P1^R), \exists v' \in Nmax(P2^R): v \in {}_{P1}{\uparrow}w \wedge v' \in {}_{P2}{\uparrow}w \wedge v = v')$

Setting $XX(P1, P2)^R := \{w \in R[A]: \exists v \in Nmax(P1^R), \exists v' \in Nmax(P2^R): v \in {}_{P1}{\uparrow}w \wedge v' \in {}_{P2}{\uparrow}w \wedge v = v'\}$

we continue: iff $w \in Nmax(P1^R) \wedge w \in Nmax(P2^R) \wedge w \in XX(P1, P2)^R$

Thus: $Nmax((P1 \blacklozenge P2)^R) = Nmax(P1^R) \cap Nmax(P2^R) \cap XX(P1, P2)^R$

Then we get: $\sigma[P1 \blacklozenge P2](R) = \{t \in R: t[A] \in max((P1 \blacklozenge P2)^R)\} =$

$\{t \in R: t[A] \in R[A] - Nmax((P1 \blacklozenge P2)^R)\} =$

$\{t \in R: t[A] \in R[A] - (Nmax(P1^R) \cap Nmax(P2^R) \cap XX(P1, P2)^R)\} =$

$\{t \in R: t[A] \in (R[A] - Nmax(P1^R)) \cup (R[A] - Nmax(P2^R)) \cup (R[A] - XX(P1, P2)^R)\} =$

$\{t \in R: t[A] \in max(P1^R) \cup max(P2^R) \cup (R[A] - XX(P1, P2)^R)\} =$

$\sigma[P1](R) \cup \sigma[P2](R) \cup \{t \in R: t[A] \in R[A] - XX(P1, P2)^R\}$

We have: $t[A] \in R[A] - XX(P1, P2)^R$ iff

$t[A] \notin XX(P1, P2)^R$ iff

$t[A] \in \{w \in R[A]: \neg(\exists v \in Nmax(P1^R), \exists v' \in Nmax(P2^R): v \in {}_{P1}{\uparrow}w \wedge v' \in {}_{P2}{\uparrow}w \wedge v = v')$ iff

$\neg(\exists v \in Nmax(P1^R), \exists v' \in Nmax(P2^R): v \in {}_{P1}{\uparrow}t[A] \wedge v' \in {}_{P2}{\uparrow}t[A] \wedge v = v')$ iff

$\neg( t[A] \in Nmax(P1^R) \cap Nmax(P2^R): {}_{P1}{\uparrow}t[A] \cap {}_{P2}{\uparrow}t[A] \neq \emptyset)$ iff

$(t[A] \in Nmax(P1^R) \cap Nmax(P2^R): {}_{P1}{\uparrow}t[A] \cap {}_{P2}{\uparrow}t[A] = \emptyset)$

Setting $YY(P1, P2)^R := \{t \in R : t[A] \in Nmax(P1^R) \cap Nmax(P2^R) \wedge {}_{P1}{\uparrow}t[A] \cap {}_{P2}{\uparrow}t[A] = \emptyset\}$

we finally get: $\sigma[P1 \blacklozenge P2](R) = \sigma[P1](R) \cup \sigma[P2](R) \cup YY(P1, P2)^R$ Q.e.d.