

Safe Reduction of Similarity Calculus to Similarity Algebra

Ingo Schmitt Nadine Schulz

Institute of Technical and Business Information Systems
Universitätsplatz 2, 39106 Magdeburg, Germany
{schmitt | nschulz}@iti.cs.uni-magdeburg.de

January 2004

Abstract

Traditional database query languages are based on set theory and crisp logic. Many applications, however, need similarity or retrieval-like queries producing results with truth values from the interval $[0, 1]$. Such truth values can be regarded as continuous membership values of tuples expressing how strongly a query is matched. Formulating queries by applying existing similarity relational algebras means to express the user's need in a procedural manner. In order to support a declarative way of formulating queries, we generalize the classical relational domain calculus by incorporating fuzzy operations and user weights. Besides defining syntax and semantics we show how to map any calculus expression onto a corresponding similarity algebra expression. In this way, we present a theoretical foundation for a declarative query language combining retrieval functionality and traditional relational databases.

1 Introduction

Queries in multimedia databases often need a combination of information retrieval mechanisms and traditional database query language constructs. Retrieval functionality is required if a query contains a similarity predicate, e.g. the query: *»Retrieve all images that are similar in form and color to the given image.«* In this example we have two conjunctively combined similarity predicates.

Traditional boolean operators, however, are not able to deal with imprecise membership data. Therefore, fuzzy-logic operations as a generalization of boolean logic are proposed [25]. Introducing user weights gives users more control to map the information need onto an adequate query. For example, we may formulate: *»Retrieve all images that are similar in form and color to the given image. And color is twice as much important as form.«*

In order to formulate queries independently from internal query processing we propose to formulate them in a declarative manner. Therefore, we enhance the classical relational domain calculus by the notion of vagueness. For query processing, calculus expressions need to be mapped to an algebra. Therefore, our similarity algebra is the target language for the mapping but not the language which the user poses queries against. Since querying directly in relation calculus is too difficult for many users we are planning to design a graphical language in a QBE-like fashion from which a mapping to our similarity calculus can be easily performed. For this reason we need a sound theoretical foundation for declarative query languages which is presented here.

Our calculus language defines core functionalities. It extends the classical relational calculus by introducing imprecise truth values in form of similarity predicates, fuzzy operators, fuzzy quantifiers, and user weights. The language is defined on the relational model, that is, imprecise data result from applying vague predicates during the query processing but not from the database directly. Since we do not specify the exact fuzzy-operations and weighting formulas our approach serves as a formal framework which can be adapted and extended to match the needs for different scenarios.

The contributions of our work can be summarized as follows:

1. We formally define a declarative query language, the similarity relational calculus, which combines the handling of imprecise truth values together with the traditional relational domain calculus.
2. The language provides operations to weight similarity predicates. Furthermore, we introduce fuzzified quantifiers.
3. We show how to map the similarity relational calculus language to a similarity algebra, because an algebra is better eligible for efficient query processing. For the mapping we distinguish between domain dependent and domain independent queries.

2 Related Work

The relational data model and its languages, the relational algebra and the relational calculus, were developed by Codd and published in [7, 8, 9, 10]. He proved the equivalence between algebra and calculus by specifying a reduction algorithm. A good overview of the theory of relational databases is given in [17, 23, 3].

An important aspect of the reduction algorithm is to restrict calculus queries so that they produce finite and domain independent results only. Such queries are called safe queries¹. Codd introduced the property of semantical safety for queries which, however, cannot be verified syntactically on a given calculus expression. Thus, [15] summarizes and proposes definitions for syntactically provable safety by exploiting the predicates *gen* and *con*. We adopt and extend this approach to verify safety syntactically.

In our approach we enhance the relational domain calculus by vague predicates. Coping with vagueness requires appropriate logical operators. Therefore, we apply techniques from fuzzy-logic [25]. Furthermore, we give users freedom to specify preferences for operands of operators. Weights on operands express their weighted contribution to the operation result. Typically, an approach as described by Fagin and Wimmers in [12] can be applied. However, we leave the weighting mechanism open. That is, our approach is not restricted to Fagin's formula. In [20] we discussed the application of Fagin's formula to complex similarity queries w.r.t. associativity and distributivity.

Another approach to specify preferences is introduced in [16]. Instead of weighting search term preferences in terms of $\gg A \text{ is better than } B \ll$ are map onto strict partial orders. Beyond it, a weighting of search terms can be indirectly expressed using certain predicates which requires the user to specify appropriate combining functions.

In the early nineties, much research was done on developing fuzzy-databases with corresponding fuzzy query languages. [4] introduces a fuzzy ER-model together with a calculus language using Fuzzy-logic. Another example is [14] which investigates how to implement a Fuzzy-SQL language on top of the commercial database system ORACLE. Most of the work in the area of fuzzy databases, however, do not support user weights. Furthermore, they rely on the two imprecise values *necessity* and *possibility* which do not conform to our intended scenario of multimedia applications.

[22] sketches the design of a fuzzy calculus, fuzzy algebra and a mapping between them. However, this work suffers from an incomplete formalization.

Most extensions of the relational model by imprecision were performed on the relational algebra, see e.g. [1, 6]. A very good work is [6] which introduced the *same*^w similarity algebra. Our proposal defines a small set of algebra operations which is powerful enough to be the target language for mapping from the similarity calculus. One problem, not considered in [6], is the observation that a weighted conjunction where the score of one operand is zero can produce a nonzero score. Furthermore, in contrast to [6] we leave the semantics of unweighted and weighted operators unspecified and require just the satisfaction of some logical rules.

An interesting approach to combine the information retrieval world with the database world is the probability relational algebra proposed in [13]. However, due to the stochastic approach they require stochastically independent events (tuples). Therefore, the authors propose intensional semantics instead of extensional semantics which is typically used in the database area. A problem with this approach occurs when an imprecise predicate violates the demand for independent events (tuples).

3 Weighting Fuzzy Operations

In general a calculus query consists of a condition X , that is composed of n predicates x_i with $i = 1 \dots n$. A complex query condition is a compound of predicates using the operators \wedge

¹[11] unveiled an inconsistency in Codd's reduction procedure concerning safety and range variables.

and \vee . Beside these connectives the negation operator is likewise important². Thus, a query condition can be defined as $X := x \mid (X \wedge \vee X) \mid \neg X \mid (X)$.

Similarity or retrieval-like predicates produce values from the interval $[0, 1]$ called scores. The overall tuple score μ based on an aggregation (conjunction, disjunction) of the specific truth values μ_i is calculated using a scoring function $S : [0, 1] \times [0, 1] \rightarrow [0, 1]$. Typically, t-norms resp. t-conorms from Fuzzy-Logic [25] are employed as scoring-functions for conjunctions resp. disjunctions. These functions must hold the following conditions:

Definition Function $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$ is a t-norm if it satisfies the criteria:

- (i) $T(\mu_1, \mu_2) = T(\mu_2, \mu_1)$ (commutativity)
- (ii) $T(\mu_1, T(\mu_2, \mu_3)) = T(T(\mu_1, \mu_2), \mu_3)$ (associativity)
- (iii) $\mu_1 \leq \mu_3 \wedge \mu_2 \leq \mu_4 \Rightarrow T(\mu_1, \mu_2) \leq T(\mu_3, \mu_4)$ (monotonicity)
- (iv) $T(\mu_1, 1) = \mu_1$ (border condition).

Definition Function $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$ is a t-conorm if it satisfies the criteria:

- (i)-(iii) same as in the definition of t-norms
- (iv) $T(\mu_1, 0) = \mu_1$ (border condition).

An aspect to enhance the flexibility to express preferences is to give users the possibility to assign weights to arguments in a compound query [6, 12, 21]. In our approach we support a binary weighting of search terms by incorporating a weight $\theta \in [0, 1]$ into the classical operators \wedge and \vee . We distinguish between left-oriented ($\overleftarrow{\wedge}_\theta, \overleftarrow{\vee}_\theta$) and right-oriented ($\overrightarrow{\wedge}_\theta, \overrightarrow{\vee}_\theta$) operators. The arrow marks the side on which the weight is stronger. Thus, $\overrightarrow{\wedge}_\theta$ denotes that the right operand is stronger weighted than the left one and $\overleftarrow{\wedge}_\theta$ denotes that the left operand is stronger weighted than right one, respectively. We require $\mu_1 \overleftarrow{\wedge}_\theta \mu_2 = \mu_1$ if $\theta = 1$ and $\mu_1 \overleftarrow{\wedge}_\theta \mu_2 = \mu_1 \wedge \mu_2$ if $\theta = 0$. The same must be hold by $\overrightarrow{\wedge}_\theta, \overleftarrow{\vee}_\theta$, and $\overrightarrow{\vee}_\theta$.

For the evaluation of weighted conjunctions and disjunctions it is necessary to incorporate the weight θ into underlying scoring functions. Therefore, in our approach the signature of a weighted scoring function, for example for $\overleftarrow{\wedge}_\theta$, is: $S_{\overleftarrow{\wedge}_\theta}^\Theta : [0, 1] \times [0, 1] \times [0, 1] \rightarrow [0, 1]$, where the first argument is given by the weight.

There exist several requirements for a weighted scoring function [12, 21, 19]. We require a weighted scoring function to be a generalization of the corresponding, unweighted scoring function. Furthermore, the weighted function should be continuous in all its arguments and in case of equal weights or equal input scores it should reduce to the unweighted scoring function.

Fagin and Wimmers proposed a weighting formula which allows to incorporate weights into any scoring function [12]. The weighting formula requires the weights to be given for each operand. Thus, a mapping of the θ value from $\overrightarrow{\wedge}_\theta$ to θ_1, θ_2 for Fagin's formula is performed by applying $\theta_1 = \frac{1-\theta}{2}$ and $\theta_2 = \frac{1+\theta}{2}$. The same holds for $\overleftarrow{\vee}_\theta$. For the left-arrowed cases the formulas for the weights θ_1 and θ_2 need to be swapped. Fagin's approach is applicable for n -ary scoring functions. Under the assumption of $\sum_i \theta_i = 1$ and $\theta_1 \geq \dots \geq \theta_n$ the weighting formula is:

$$S^\Theta(\mu_1, \dots, \mu_n, \theta_1, \dots, \theta_n) = (\theta_1 - \theta_2)S(\mu_1) + 2 * (\theta_2 - \theta_3)S(\mu_1, \mu_2) + \dots + n * \theta_n S(\mu_1, \dots, \mu_n)$$

²In this section quantifiers are neglected.

where S^\ominus denotes the weighted scoring function based on the unweighted function S , the weights θ_i , and the scores μ_i with $i = 1, \dots, n$. There exist further weighted scoring functions employed in various retrieval systems, see e. g. in [5, 18]. Nevertheless, we recommend to use Fagin's weighting formula to obtain weighted scoring functions since these meet most of our requirements.

One problem occurs if Fagin's formula is applied to a scoring function without idempotence. In case of equal input scores but different weights the formula does not reduce to the unweighted case. Assume, for example, the binary t-norm be the algebraic product $\mu_1 * \mu_2$ and $\mu_1 = \mu_2$ then Fagin's weighting produces

$$(\theta_1 - \theta_2) * \mu_1 + 2 * \theta_2 * \mu_1 * \mu_1 = (1 - 2\theta_2) * \mu_1 + 2 * \theta_2 * \mu_1 * \mu_1 \neq \mu_1 * \mu_1.$$

We suggest a small modification of Fagin's formula:

$$S^\ominus(\mu_1, \mu_2, \theta_1, \theta_2) = (\theta_1 - \theta_2)S(\mu_1, \mu_1) + 2 * \theta_2 * S(\mu_1, \mu_2)$$

to solve this problem.

For the evaluation of a universal combination of truth values fuzzy compensatory operators can be applied. In order to achieve a compensator effect between the scores μ_1 and μ_2 of an object regarding the conditions x_1 and x_2 an intersection-operator (f_\wedge) is combined with a union-operator (f_\vee). Typically, t-norms and t-conorms are applied. The combination results from a linear (equation 1) or an exponential combination (equation 2) of both operations [26].

$$f(\mu_1, \mu_2) = \gamma * f_\vee(\mu_1, \mu_2) + (1 - \gamma) * f_\wedge(\mu_1, \mu_2) \quad (1)$$

$$f(\mu_1, \mu_2) = f_\vee(\mu_1, \mu_2)^\gamma * f_\wedge(\mu_1, \mu_2)^{1-\gamma} \quad (2)$$

The behavior of the compensatory operator can be modified by the parameter $\gamma \in [0, 1]$. With $\gamma = 0$ the operator corresponds to f_\wedge and with $\gamma = 1$ the operator results in f_\vee .

In our approach we use for the evaluation of a universal combination the fuzzy compensatory OR-operator. This operator is defined as follows:

$$f(\mu_1, \mu_2) = z * \min(\mu_1, \mu_2) + (1 - z) * \max(\mu_1, \mu_2)$$

with $z \in [0, 1]$ [24]. The compensatory OR-operator results in the t-norm \min if $z = 1$ and in the t-conorm \max if $z = 0$, respectively. Instead of using \min and \max any other t-norm and t-conorm can be applied.

4 Similarity Calculus

In this section we formally define the syntax and semantics of our similarity calculus language. The design is based on the following principles:

1. The language is a generalization of the relational domain calculus. Thus, every traditional relational domain query can be expressed and evaluated in our language producing the same query result.
2. Fuzzy truth values are generated primarily by applying similarity predicates.

3. The result of a query is a relation which contains all tuples with a non-zero truth value.
4. Our language definitions provide an open framework for different scenarios with corresponding similarity predicates. That is, we exactly define the semantics of many language constructs, but leave the semantics of similarity predicates open. In this way, our language is open to cope, for example, with histogram intersections as a special similarity predicate. This predicate is required for measuring image similarity upon color distribution.
5. User weights on operators are expressed by weight variables. Their values are fixed outside the query by an interpretation function.
6. We introduce fuzzyfied quantifiers which soften the strict semantics of traditional quantifiers.

4.1 Syntax

We start by defining the basic syntax elements of the language.

Definition We denote the *similarity domain relation calculus* SDC as a tuple $(\mathbf{U}, \mathbf{X}, \Delta, \mathbf{C}, \mathbf{D}, Dom, \mathbf{R}, \Theta, \mathbf{Z})$, where $\mathbf{U} = \{A_1, A_2, \dots\}$ is the universe of attributes; $\mathbf{X} = \{X_1, X_2, \dots\}$ is a set of variables; $\Delta = \{\delta_1, \delta_2, \dots\}$ is a set of binary, typed³ operation names (we distinguish continuous operations, e.g. similarity operations, from discrete operations, e.g. traditional comparison operators like $<, \leq, =, \neq, >, \geq$); \mathbf{C} is a set of constant names; \mathbf{D} is a set of domain names; Dom is a mapping from $\mathbf{U} \cup \mathbf{X} \cup \Delta \cup \mathbf{C}$ to \mathbf{D} ; \mathbf{R} is a finite set of relation schemata R_1, R_2, \dots, R_p , all are subsets⁴ of \mathbf{U} ; $\Theta = \{\theta_1, \theta_2, \dots\}$ is a set of weight variables; and $\mathbf{Z} = \{z_1, z_2, \dots\}$ is a set of operator weight variables.

We build a calculus query expression E over SDC from *atoms* and *formulas*.

Definition Let an *atom* be

1. $R(Y_1, Y_2, \dots, Y_m)$, where $R \in \mathbf{R}$ is a relation schema A_1, A_2, \dots, A_m and for each $Y_i \in \mathbf{X} \cup \mathbf{C}$ the domain is sound: $Dom(Y_i) = Dom(A_i)$.
2. $Y_1 \delta Y_2$, where $\delta \in \Delta$ is an operation name (infix notation) and $Y_1, Y_2 \in \mathbf{X} \cup \mathbf{C}$ are consistently typed ($Dom(Y_1) = Dom(Y_2) = Dom(\delta)$).

Please note, that both operands of a binary operation can be constants at the same time. In this case, interpreting such an atom results in a truth value independent from any variable.

Definition An SDC -*formula* $F(X_1, X_2, \dots, X_n)$ ⁵, where $X_i \in \mathbf{X}, i = 1, \dots, n$ are the involved free variables, is recursively defined:

1. Any atom is a formula $F(X_1, X_2, \dots, X_n)$ where X_i are the involved variables.

³Both operands are assumed to be from the same domain.

⁴For convenience, we assume a fixed attribute order.

⁵As short form we often write F instead of $F(X_1, X_2, \dots, X_n)$.

2. $(F_a(X_{a_1}, \dots, X_{a_k}) \phi F_b(X_{b_1}, \dots, X_{b_l}))$ with $\phi \in \{\wedge, \overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \vee, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta, \oplus^z, \overleftarrow{\oplus}_\theta, \overrightarrow{\oplus}_\theta\}$ is a formula $F(X_1, \dots, X_n)$ if $F_a(X_{a_1}, \dots, X_{a_k})$ as well as $F_b(X_{b_1}, \dots, X_{b_l})$ are formulas. Involved variables are united: $\{X_1, \dots, X_n\} = \{X_{a_1}, \dots, X_{a_k}\} \cup \{X_{b_1}, \dots, X_{b_l}\}$.

In case of using weighted operators $(\overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta, \overleftarrow{\oplus}_\theta, \overrightarrow{\oplus}_\theta)$, $\theta \in \Theta$ is a weight variable and $\overrightarrow{\wedge}_\theta$ resp. $\overrightarrow{\vee}_\theta, \overleftarrow{\oplus}_\theta$ denotes that F_b is stronger weighted than F_a and $\overleftarrow{\wedge}_\theta$ resp. $\overleftarrow{\vee}_\theta, \overrightarrow{\oplus}_\theta$ denotes that F_a is stronger weighted than F_b , respectively. The universal operators $(\overleftarrow{\oplus}^z, \overleftarrow{\oplus}_\theta, \overrightarrow{\oplus}_\theta)$ can be parameterized using the operator weight variable $z \in \mathbf{Z}$.

3. $(\neg F(X_1, \dots, X_n))$ is a formula if $F(X_1, \dots, X_n)$ is a formula.
4. $(\exists X F(X_1, \dots, X_n))$ is a formula if $F(X_1, \dots, X_n)$ is a formula and $X \in \{X_1, \dots, X_n\}$.
5. $(\forall X F(X_1, \dots, X_n))$ is a formula if $F(X_1, \dots, X_n)$ is a formula and $X \in \{X_1, \dots, X_n\}$.
6. $(\exists_k X F(X_1, \dots, X_n))$ is a formula if $F(X_1, \dots, X_n)$ is a formula, $k > 1$ is a natural number, and $X \in \{X_1, \dots, X_n\}$.
7. $(\forall_k X F(X_1, \dots, X_n))$ is a formula if $F(X_1, \dots, X_n)$ is a formula, $k > 1$ is a natural number, and $X \in \{X_1, \dots, X_n\}$.
8. $(\overset{\theta}{\succ} F(X_1, \dots, X_n))$ is a formula if $F(X_1, \dots, X_n)$ is a formula. This cut-operator works similar to a trigger w.r.t. the threshold θ . Any fuzzy value above or equal the threshold produces the value one otherwise the value zero.
9. $(\overset{\theta}{\prec} F(X_1, \dots, X_n))$ is a formula if $F(X_1, \dots, X_n)$ is a formula. This softcut-operator is a modification of the cut-operator, which sets the fuzzy value to zero if it is below the threshold value. Otherwise the fuzzy-value remains unchanged.

In our language we distinguish between left-oriented $(\overleftarrow{\wedge}_\theta, \overleftarrow{\vee}_\theta)$ and right-oriented $(\overrightarrow{\wedge}_\theta, \overrightarrow{\vee}_\theta)$ weighted operators. This is necessary because a weighted conjunction can produce a membership value of non-zero although one operand equals zero. The reason is the requirement for a weighting formula, that if a weight is completely on one side, the other operand should be completely ignored. In order to reason over weighted operators we introduce, therefore, asymmetric weighted operators. As later specified, this distinction is also important for generating the predicates *gen* and *con* required for guaranteeing safe queries.

The normal exists- and forall-quantifier are sometimes too strict because their results often depend on one single value. Therefore, we introduce the fuzzyfied quantifiers $\exists_k X$ and $\forall_k X$, also called *few* and *most*. In our approach, these quantifiers need at least k different significant X -values to behave like the normal \exists and \forall , respectively. Otherwise, the quantifiers provide only a corresponding fraction of the normal quantifier value. The formal definition is given in Definition 4.2.

Definition A query expression E over SDC has the form

$$\{X_1, X_2, \dots, X_n | F(X_1, X_2, \dots, X_n)\}$$

where $F(X_1, X_2, \dots, X_n)$ is an SDC-formula and X_1, \dots, X_n are the involved free variables.

The query asks for the values for all variables where the condition F holds.

In order to guarantee finite and domain independent results we require safe queries. To verify safety syntactically we adopt and extend the approach given in [15]. We require a query to be evaluable since evaluable formulas comprise the largest class of domain independent formulas which can be tested syntactically. The test on safety is based on the special predicates **gen** and **con** defined in the Definitions 4.1 and 4.1.

Definition An *SDC* query $\{X_1, X_2, \dots, X_n | F(X_1, X_2, \dots, X_n)\}$ is *evaluable* if

1. for every variable X_i with $i = 1, \dots, n$ exists a relational algebra expression E where the fact $gen(X_i, F, E)$ is defined,
2. for every variable X bound in a subformula $(\exists X F_1)$ of F exists a relational algebra expression E where the fact $con(X, F_1, E)$ is defined,
3. for every variable X bound in a subformula $(\forall X F_1)$ of F exists a relational algebra expression E where the fact $con(X, \neg F_1, E)$ is defined, and
4. for every variable X bound in a subformula $(\exists_k X F_1)$ of F exists a relational algebra expression E where the fact $con(X, F_1, E)$ is defined and E does not contain the symbol ' \perp ',
5. for every variable X bound in a subformula $(\forall_k X F_1)$ of F exists a relational algebra expression E where the fact $con(X, \neg F_1, E)$ is defined and E does not contain the symbol ' \perp ',
6. $F(X_1, X_2, \dots, X_n)$ is rectified⁶.

We discuss the motivation behind rule 4 and 5 of Definition 4.1 in Section 7.

Regarding safety of a formula we are interested to know whether every value of a variable X resulting from formula F is a value from the finite database or not. This information can be recursively induced by the relation **gen**. If a relational expression E over the database is assigned to a variable and a formula via **gen**, then all satisfying variable values for X are values contained in the evaluation result of E .

Definition Let the predicate $gen(X, F, E)$, where $X \in \mathbf{X}$ is a variable, F an *SDC*-formula, and E is an relational algebra expression be inductively defined:

$$\begin{aligned}
 gen(X, F, \pi_{\#i}(R))^7 & \text{ if } F = R(Y_1, \dots, Y_{i-1}, X, Y_{i+1}, \dots, Y_m). \\
 gen(X, (F_a \wedge F_b), E) & \text{ if } gen(X, F_a, E). \\
 gen(X, (F_a \wedge F_b), E) & \text{ if } gen(X, F_b, E). \\
 gen(X, (F_a \overleftarrow{\wedge}_\theta F_b), E) & \text{ if } gen(X, F_a, E). \\
 gen(X, (F_a \overrightarrow{\wedge}_\theta F_b), E) & \text{ if } gen(X, F_b, E). \\
 gen(X, (F_a \phi F_b), (E_a) \cup (E_b)) & \text{ if } gen(X, F_a, E_a) \text{ and } gen(X, F_b, E_b). \\
 gen(X, (\neg F), E) & \text{ if } gen(X, pushnot(\neg F), E). \\
 gen(X, (\exists X_1 F), E) & \text{ if } gen(X, F, E) \text{ and } X \neq X_1.
 \end{aligned}$$

⁶A formula is called rectified if (1) no variable occurs both bound and free, and all quantifiers in the formula refer to different variables.

$$\begin{aligned}
 \text{gen}(X, (\forall X_1 F), E) & \text{ if } \text{gen}(X, F, E) \text{ and } X \neq X_1. \\
 \text{gen}(X, (\exists_k X_1 F), E) & \text{ if } \text{gen}(X, F, E) \text{ and } X \neq X_1. \\
 \text{gen}(X, (\forall_k X_1 F), E) & \text{ if } \text{gen}(X, F, E) \text{ and } X \neq X_1. \\
 \text{gen}(X, (\overset{\theta}{\succ} F), E) & \text{ if } \text{gen}(X, F, E). \\
 \text{gen}(X, (\overset{\theta}{\prec} F), E) & \text{ if } \text{gen}(X, F, E).
 \end{aligned}$$

The symbol ϕ stands for any of the six operators $\vee, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta, \oplus^z, \overset{\leftarrow z}{\oplus}_\theta, \overset{\rightarrow z}{\oplus}_\theta$. The operation *pushnot* pushes syntactically the negation within a formula to the atoms by applying DeMorgan's laws on $\vee, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta, \oplus^z, \overset{\leftarrow z}{\oplus}_\theta, \overset{\rightarrow z}{\oplus}_\theta, \wedge, \overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta$, involution, $(\neg(\exists X F)) \Rightarrow (\forall X(\neg F))$, $(\neg(\forall X F)) \Rightarrow (\exists X(\neg F))$, $(\neg(\exists_k X F)) \Rightarrow (\forall_k X(\neg F))$, $(\neg(\forall_k X F)) \Rightarrow (\exists_k X(\neg F))$ ⁸, $(\neg \overset{\theta}{\succ} F) \Rightarrow (\overset{1-\theta}{\succ} \neg F)$ ⁹, and $(\neg \overset{\theta}{\prec} F) \Rightarrow (\neg((\overset{\theta}{\succ} F) \wedge F))$.

Whereas the relation **gen** is used for testing safety of unbound variables, the relation **con** is designed to test safety of bound variables. Since values of a bound variable do not appear in the calculus expression result, we have just to guarantee, that the evaluation of quantifiers can be performed on finite sets. This is true if either every value satisfying the subformula is a database value or a subformula does not contain the bound variable. In the latter case, denoted by the symbol ' \perp ', the evaluation does not require any value for the bound variable.

Definition Let the predicate $\text{con}(X, F, E)$, where $X \in \mathbf{X}$ is a variable, F an *SDC*-formula, and E is an algebra expression be inductively defined:

$$\begin{aligned}
 \text{con}(X, F, \pi_{\#_i}(R)) & \text{ if } F = R(Y_1, \dots, Y_{i-1}, X, Y_{i+1}, \dots, Y_m). \\
 \text{con}(X, F(X_1, \dots, X_m), \perp) & \text{ if } X_i \neq X \text{ for } i = 1, \dots, m \\
 \text{con}(X, (F_a \wedge F_b), E) & \text{ if } \text{gen}(X, F_a, E). \\
 \text{con}(X, (F_a \wedge F_b), E) & \text{ if } \text{gen}(X, F_b, E). \\
 \text{con}(X, (F_a \overleftarrow{\wedge}_\theta F_b), E) & \text{ if } \text{gen}(X, F_a, E). \\
 \text{con}(X, (F_a \overrightarrow{\wedge}_\theta F_b), E) & \text{ if } \text{gen}(X, F_b, E). \\
 \text{con}(X, (F_a \phi F_b), (E_a) \cup (E_b)) & \text{ if } \text{con}(X, F_a, E_a) \text{ and } \text{con}(X, F_b, E_b). \\
 \text{con}(X, (\neg F), E) & \text{ if } \text{con}(X, \text{pushnot}(\neg F), E). \\
 \text{con}(X, (\exists X_1 F), E) & \text{ if } \text{con}(X, F, E) \text{ and } X \neq X_1. \\
 \text{con}(X, (\forall X_1 F), E) & \text{ if } \text{con}(X, F, E) \text{ and } X \neq X_1. \\
 \text{con}(X, (\exists_k X_1 F), E) & \text{ if } \text{con}(X, F, E) \text{ and } X \neq X_1. \\
 \text{con}(X, (\forall_k X_1 F), E) & \text{ if } \text{con}(X, F, E) \text{ and } X \neq X_1. \\
 \text{con}(X, (\overset{\theta}{\succ} F), E) & \text{ if } \text{con}(X, F, E). \\
 \text{con}(X, (\overset{\theta}{\prec} F), E) & \text{ if } \text{con}(X, F, E).
 \end{aligned}$$

The symbol ' \perp ' stands for the empty set and $\phi \in \{\wedge, \overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \vee, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta, \oplus^z, \overset{\leftarrow z}{\oplus}_\theta, \overset{\rightarrow z}{\oplus}_\theta\}$ holds. Thus, ' \perp ' is the neutral element w.r.t. to the union.

⁷The projection in the relational algebra expression identifies attributes by their position, that is, $\#_i$ refers to the i -th attribute.

⁸Involution as well as DeMorgan's laws are required for weighted operators by Definition 4.2.

⁹Since pushnot works completely syntactically we can ignore the asymmetry of the threshold.

4.2 Semantics

After defining the syntax of our \mathcal{SDC} language we specify the semantics of \mathcal{SDC} -expressions. Therefore, we first give the interpretation over \mathcal{SDC} in Definition 4.2 followed by definitions for variable mapping and evaluating atoms. We specify the semantics of an \mathcal{SDC} -formula in Definition 4.2 and, finally, we define the semantics of an \mathcal{SDC} -query expression.

Definition An *interpretation* over $\mathcal{SDC}(\mathbf{U}, \mathbf{X}, \Delta, \mathbf{C}, \mathbf{D}, \text{Dom}, \mathbf{R}, \Theta, \mathbf{Z})$ is a triple $(\mathbf{d}, \mathbf{db}, I)$

1. \mathbf{d} is a finite set of domains $\{d_1, d_2, \dots, d_q\}$, each domain is a non-empty, not necessarily finite set of values and \mathbf{db} is a finite set of finite relations $\{r_1, r_2, \dots, r_p\}$ over these domains.

2. I is an interpretation function which

- a) maps any domain name $D \in \mathbf{D}$ to a domain

$$I(D) \in \mathbf{d},$$

- b) maps any relation schema $R(A_1, A_2, \dots, A_n) \in \mathbf{R}$ to a relation

$$I(R) \in \mathbf{db},$$

where $I(R) \subseteq I(\text{Dom}(A_1)) \times I(\text{Dom}(A_2)) \times \dots \times I(\text{Dom}(A_n))$,

- c) maps any operation name $\delta \in \Delta$ to a binary function:

$$I(\delta) : I(\text{Dom}(\delta)) \times I(\text{Dom}(\delta)) \rightarrow [0, 1].$$

¹⁰ By convention, there exists an operation $,='$ with the equality-semantics which can be applied to every datatype.

- d) maps any constant name $C \in \mathbf{C}$ to a value

$$I(C) \in I(\text{Dom}(C)),$$

- e) maps any weight variable $\theta \in \Theta$ to a value

$$I(\theta) \in [0, 1],$$

- f) maps any operator weight variable $z \in \mathbf{Z}$ to a value

$$I(z) \in [0, 1],$$

- g) maps the conjunction symbol $'\wedge'$ to a fuzzy t-norm

$$I(\wedge) : [0, 1] \times [0, 1] \rightarrow [0, 1].$$

It must hold:

$$\forall \mu \in [0, 1] : I(\wedge)(0, \mu) = I(\wedge)(\mu, 0) = 0.$$

Since $'\wedge'$ is associative and commutative we generalize it to an n -ary operator.

¹⁰Discrete operation values are restricted to $\{0, 1\}$ where 0 denotes **false** and 1 denotes **true**.

h) maps the weighted conjunction symbol ' $\vec{\wedge}_\theta$ ' to a weighted fuzzy t-norm

$$I(\vec{\wedge}_\theta) : [0, 1] \times [0, 1] \times [0, 1] \rightarrow [0, 1].$$

The first parameter is reserved for the weight $I(\theta)$. Furthermore, the following condition must hold:

$$\forall v, \mu \in [0, 1], I(\vec{\wedge}_\theta)(v, \mu, 0) = 0.$$

The value 0 for θ produces the unweighted conjunction whereas the value 1 ignores the less weighted operand:

$$\begin{aligned} \forall \mu_1, \mu_2 \in [0, 1] : I(\vec{\wedge}_\theta)(0, \mu_1, \mu_2) &= I(\wedge)(\mu_1, \mu_2), \\ \forall \mu_1, \mu_2 \in [0, 1] : I(\vec{\wedge}_\theta)(1, \mu_1, \mu_2) &= \mu_2. \end{aligned}$$

Furthermore, equal input scores are reduced to the unweighted case:

$$\forall \theta, \mu \in [0, 1] : I(\vec{\wedge}_\theta)(\theta, \mu, \mu) = I(\wedge)(\mu, \mu)$$

Mapping the weighted conjunction symbol ' $\overleftarrow{\wedge}_\theta$ ' is analogously.

i) maps the disjunction symbol ' \vee ' to a fuzzy t-conorm

$$I(\vee) : [0, 1] \times [0, 1] \rightarrow [0, 1].$$

It must hold:

$$\forall \mu \in [0, 1] : I(\vee)(1, \mu) = I(\vee)(\mu, 1) = 1.$$

Since ' \vee ' is associative and commutative we generalize it to an n -ary operator.

j) maps the weighted disjunction symbol ' $\vec{\vee}_\theta$ ' to a weighted fuzzy t-conorm

$$I(\vec{\vee}_\theta) : [0, 1] \times [0, 1] \times [0, 1] \rightarrow [0, 1].$$

The first parameter is reserved for the weight $I(\theta)$. Furthermore, the following condition must hold:

$$\forall v, \mu \in [0, 1], I(\vec{\vee}_\theta)(v, \mu, 1) = 1.$$

The value 0 for θ produces the unweighted disjunction whereas the value 1 ignores the less weighted operand:

$$\begin{aligned} \forall \mu_1, \mu_2 \in [0, 1] : I(\vec{\vee}_\theta)(0, \mu_1, \mu_2) &= I(\vee)(\mu_1, \mu_2), \\ \forall \mu_1, \mu_2 \in [0, 1] : I(\vec{\vee}_\theta)(1, \mu_1, \mu_2) &= \mu_2. \end{aligned}$$

Furthermore, equal input scores are reduced to the unweighted case:

$$\forall \theta, \mu \in [0, 1] : I(\vec{\vee}_\theta)(\theta, \mu, \mu) = I(\vee)(\mu, \mu)$$

Mapping the weighted disjunction symbol ' $\overleftarrow{\vee}_\theta$ ' is analogously.

k) maps the universal combination symbol ‘ \oplus^z ’ to a fuzzy compensatory OR-operator

$$I(\oplus^z) : [0, 1] \times [0, 1] \times [0, 1] \rightarrow [0, 1].$$

The first parameter is reserved for the operator weight $I(z)$. Due to the operator weight ‘ \oplus^z ’ is not associative. The following condition must hold:

$$\forall \mu_1, \mu_2 \in [0, 1], I(\oplus^z)(0, \mu_1, \mu_2) = I(\vee)(\mu_1, \mu_2).$$

l) maps the weighted universal combination symbol ‘ $\overrightarrow{\oplus}_\theta^z$ ’ to a weighted fuzzy compensatory OR-operator

$$I(\overrightarrow{\oplus}_\theta^z) : [0, 1] \times [0, 1] \times [0, 1] \times [0, 1] \rightarrow [0, 1].$$

The first parameter is reserved for the operator weight $I(z)$ and the second one for $I(\theta)$. Furthermore, the following condition must hold:

$$\forall v, \mu_1, \mu_2 \in [0, 1], I(\overrightarrow{\oplus}_\theta^z)(0, v, \mu, \mu_2) = I(\overrightarrow{\vee}_\theta)(\mu_1, \mu_2).$$

The value 0 for θ produces the unweighted universal combination whereas the value 1 ignores the less weighted operand:

$$\begin{aligned} \forall z \in [0, 1], \mu_1, \mu_2 \in [0, 1] : I(\overrightarrow{\oplus}_\theta^z)(z, 0, \mu_1, \mu_2) &= I(\oplus^z)(z, \mu_1, \mu_2), \\ \forall z \in [0, 1], \mu_1, \mu_2 \in [0, 1] : I(\overrightarrow{\oplus}_\theta^z)(z, 1, \mu_1, \mu_2) &= \mu_2. \end{aligned}$$

Mapping the weighted universal combination symbol ‘ $\overleftarrow{\oplus}_\theta^z$ ’ is analogously.

m) maps the negation symbol ‘ \neg ’ to a fuzzy negation:

$$I(\neg) : [0, 1] \rightarrow [0, 1] \text{ with } \neg\neg\mu = \mu.$$

Furthermore, we require the fuzzy negation to conform weighted and unweighted disjunction/conjunction w.r.t. DeMorgan’s laws.

Please notice, the semantics of the operations $\delta, \wedge, \overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \vee, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta, \oplus^z, \overrightarrow{\oplus}_\theta^z, \overleftarrow{\oplus}_\theta^z$ is not predefined and can therefore be arbitrarily defined as long as the specified restrictions are met. In this way, the language is defined as a framework which works with many domain specific similarity operations and fuzzy operations.

The common semantics of the fuzzy-conjunction is the **min**-function, and the **max**-function for the fuzzy-disjunction. The dominant weighting formula is Fagin’s formula described in [12].

In the next definition we assign a value to every variable.

Definition Let V be a *variable mapping* from \mathbf{X} to $\bigcup_{d \in \mathbf{d}} d$ where $\forall X \in \mathbf{X}. V(X) \in I(\text{Dom}(X))$ holds.

Now we can assign a value to every atom.

Definition The *evaluation* $V^*(F)$ of an *SDC* atom F with respect to a variable mapping V and an interpretation function I is given by:

1. If $F = R(Y_1, Y_2, \dots, Y_m)$ then $V^*(F) = 1$ if

$$(v_1, v_2, \dots, v_m) \in I(R)$$

where

$$i \in \{1, \dots, m\}.v_i = \begin{cases} V(Y_i) & \text{if } Y_i \text{ is a variable} \\ I(Y_i) & \text{if } Y_i \text{ is a constant name,} \end{cases}$$

otherwise $V^*(F) = 0$.

2. If $F = Y_1 \delta Y_2$ then $V^*(F) = I(\delta)(v_1, v_2)$ where v_i is defined as above.

Definition The *semantics* of an \mathcal{SDC} -formula $F(X_1, X_2, \dots, X_n)$ denoted $I_V^*(F)$ basing on the evaluation of atoms $V^*(F)$ and an interpretation function I is recursively defined:

1. If F is an atom then

$$I_V^*(F) = V^*(F).$$

2. If F is a conjunction ($F_1 \wedge F_2$) then

$$I_V^*(F) = I(\wedge)(I_V^*(F_1), I_V^*(F_2)).$$

3. If F is a disjunction ($F_1 \vee F_2$) then

$$I_V^*(F) = I(\vee)(I_V^*(F_1), I_V^*(F_2)).$$

4. If F is a weighted conjunction or disjunction ($F_1 \phi F_2$) with $\phi \in \{\vec{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \vec{\vee}_\theta, \overleftarrow{\vee}_\theta\}$ then

$$I_V^*(F) = I(\phi)(I(\theta), I_V^*(F_1), I_V^*(F_2)).$$

5. If F is a universal combination ($F_1 \oplus^z F_2$) then

$$I_V^*(F) = I(\oplus^z)(I(z), I_V^*(F_1), I_V^*(F_2)).$$

6. If F is a weighted universal combination ($F_1 \phi F_2$) and $\phi \in \{\overleftarrow{\oplus}_\theta, \overrightarrow{\oplus}_\theta\}$ then

$$I_V^*(F) = I(\phi)(I(z), I(\theta), I_V^*(F_1), I_V^*(F_2)).$$

7. If F is a negation ($\neg F_1$) then

$$I_V^*(F) = I(\neg)(I_V^*(F_1)).$$

8. If F is an existentially bound formula ($\exists X F_1(X_1, \dots, X_n)$) then

$$I_V^*(F) = I(\vee)(I_{\mathbf{V}_X}^*(F_1))$$

where $\mathbf{V}_X = \{V_X \text{ is a variable mapping where } X_i \neq X \text{ implies } V_X(X_i) = V(X_i) \text{ for all } X_i \in \mathbf{X}\}$. The expression $I(\vee)(I_{\mathbf{V}_X}^*(F_1))$ denotes a disjunction over all variable mappings from \mathbf{V}_X applied to F_1 . The set \mathbf{V}_X can be infinite, if a query is not evaluable. Due to distributivity and associativity, the disjunction can be applied to an arbitrary number of truth values. Applied to one value it returns exactly that value.

-
9. The \forall -case is analogous to the \exists -case except the disjunction is replaced by the conjunction.
10. If F is a k -existentially bound formula $(\exists_k X F_1(X_1, \dots, X_n))$ then

$$I_V^*(F) = I(\vee)(I_{\mathbf{V}_X}^*(F_1)) * \min \left(\frac{\sum_{V_X \in \mathbf{V}_X} I_{V_X}^*(F_1)}{k * I(\vee)(I_{\mathbf{V}_X}^*(F_1))}, 1 \right).$$

11. If F is a k -universally bound formula $(\forall_k X F_1(X_1, \dots, X_n))$ then

$$I_V^*(F) = I_V^*((\neg(\exists_k(\neg F_1(X_1, \dots, X_n))))).$$

12. If F is a cut-operation $(\overset{\theta}{\succ} F_1)$ then

$$I_V^*(F) = \begin{cases} 1 & \text{if } I_V^*(F_1) \geq I(\theta) \\ 0 & \text{otherwise} \end{cases}$$

13. If F is a softcut-operation $(\overset{\theta}{\succ} F_1)$ then

$$I_V^*(F) = I_V^*((\overset{\theta}{\succ} F_1) \wedge F_1)$$

Now we are able to define the semantics of a query expression. We require a non-zero truth value for every tuple of the result. Thus, the result of a query is a relation.

Definition The *semantics* of a query expression $E = \{X_1, X_2, \dots, X_n | F(X_1, X_2, \dots, X_n)\}$ denoted $I^*(E)$ is $\{(V(X_1), V(X_2), \dots, V(X_n)) | I_V^*(F(X_1, X_2, \dots, X_n)) > 0\}$.

5 Similarity Algebra

In the following we introduce our similarity algebra \mathcal{SA} , which is the target language for mapping from \mathcal{SDC} expressions. We do not consider it as a language where user formulates queries against. \mathcal{SA} enhances traditional relational algebra by introducing vagueness and weighting. The similarity algebra \mathcal{SA} is defined as follows:

Definition The tuple $(\mathbf{U}, \Delta, \mathbf{C}, \mathbf{D}, \text{Dom}, \mathbf{R}, \Theta, \mathbf{Z})$ denotes the *similarity algebra* \mathcal{SA} , where $\mathbf{U}, \Delta, \mathbf{C}, \mathbf{D}, \text{Dom}, \mathbf{R}, \Theta, \mathbf{Z}$ have the same meaning as for the \mathcal{SDC} language (see Definition 4.1).

Let $\text{att}(E)$ be all attributes occurring in an algebra expression E . Every attribute is denoted by the ordinal number $\#_i$ of its occurrence in E . We now define a similarity algebra expression as given in the following definition.

Definition A *similarity algebra expression* E over a similarity algebra \mathcal{SA} is the smallest class of expressions which include the following:

1. $\mathbf{0}$, that is a special relation needed for the mapping.

2. $\mathbf{1}$, that is a special relation needed for the mapping.
3. $R \in \mathbf{R}$
4. Dom_D with $D \in \mathbf{D}$
5. $\pi_{\#_{p_1}, \#_{p_2}, \dots, \#_{p_n}}(E)$, where E is a similarity algebra expression and $\{\#_{p_1}, \#_{p_2}, \dots, \#_{p_n}\} \subseteq att(E)$.
6. $\pi_{\#_{p_1}, \#_{p_2}, \dots, \#_{p_n}}^k(E)$, where E is a similarity algebra expression, $k > 1$ a natural number, and $\{\#_{p_1}, \#_{p_2}, \dots, \#_{p_n}\} \subseteq att(E)$. This operation is the counterpart for the k -exists-quantifier of the similarity calculus.
7. $\sigma_{y_i \delta y_j}(E)$, where $\delta \in \Delta$ is a binary operation and $Dom(y_i) = Dom(y_j) = Dom(\delta)$, $\{y_i, y_j\} \subseteq att(E) \cup \mathbf{C}$, and E is a similarity algebra expression. Both operands can be constants at the same time.

We sometimes specify a list of conjunctively combined simple conditions as a short form of a list of corresponding selections.

8. unions: $(E_1 \cup E_2)$, $(E_1 \vec{\cup}_\theta E_2)$, $(E_1 \overleftarrow{\cup}_\theta E_2)$ where E_1 and E_2 union compatible¹¹ similarity algebra expressions.
9. intersections: $(E_1 \cap E_2)$, $(E_1 \vec{\cap}_\theta E_2)$, $(E_1 \overleftarrow{\cap}_\theta E_2)$ where E_1 and E_2 are union compatible similarity algebra expressions.
10. universal combinations: $(E_1 \uplus^z E_2)$, $(E_1 \overrightarrow{\uplus}_\theta^z E_2)$, $(E_1 \overleftarrow{\uplus}_\theta^z E_2)$ where E_1 and E_2 are union compatible similarity algebra expressions.
11. $(E_1 \setminus E_2)$, where E_1 and E_2 are union compatible similarity algebra expressions.
12. $(E_1 \times E_2)$, where E_1 and E_2 are similarity algebra expressions. Since ‘ \times ’ is associative and commutative we generalize it to an n -ary cartesian product.
13. $(E_1 \bowtie_{\#_{i_1}=\#_{j_1}, \dots, \#_{i_m}=\#_{j_m}} E_2)$ where E_1 and E_2 are similarity algebra expressions and $\{\#_{i_1}, \dots, \#_{i_m}\} \subseteq att(E_1)$, $\{\#_{j_1}, \dots, \#_{j_m}\} \subseteq att(E_2)$.
14. $\overset{\theta}{\succ} (E)$ where E is a similarity algebra expression.
15. $\overset{\theta}{\prec} (E)$ where E is a similarity algebra expression.

The *interpretation* of a similarity algebra $SA(\mathbf{U}, \Delta, \mathbf{C}, \mathbf{D}, Dom, \mathbf{R}, \Theta)$ except for the operations is the same as for \mathcal{SDC} . Thus, it is a triple $(\mathbf{d}, \mathbf{db}, I)$, too. Therefore, we refer to Definition 4.2.

In the following, we define the semantics of an algebra expression. Notice, that we equip relations with an artificial membership attribute $\#_0$ at first attribute position. Our algebra can deal with infinite sets. Since the algebra is intended to be a target language we assume any algebra expression to be created by mapping an \mathcal{SDC} -query. As we will see later, any evaluable \mathcal{SDC} -query is mapped to an algebra expression basing on finite sets.

¹¹Union compatibility means that the two expressions E_1 and E_2 must share the same number of attributes and the same domain for every corresponding attribute pair.

Definition The *semantics* of an algebra expression E is inductively defined by the interpretation function I^* :

1. *0-relation* $E = \mathbf{0}$: $I^*(\mathbf{0}) = \{(1, 0)\}$ is a relation with exactly one tuple with membership value 1 and an arbitrary attribute value preferable the value 0.
2. *1-relation* $E = \mathbf{1}$: $I^*(\mathbf{1}) = \{(1)\}$ is a relation with exactly one tuple with membership value 1 and no attribute value.
3. *relation name* $E = R \in \mathbf{R}$: $I^*(R) = \{(1, v_1, v_2, \dots, v_n) | (v_1, v_2, \dots, v_n) \in I(R)\}$ where A_1, A_2, \dots, A_n are the attributes of R . All tuples are equipped with a membership value 1 since they are considered as true facts.
4. *domain* $E = Dom_D$: $I^*(Dom_D) = \{(1, v) | v \in I(D)\}$. All domain values are equipped with a membership value 1 since they are considered as true facts. Please notice, that the interpretation of a domain can be an infinite set.
5. *projection* $E = \pi_{\#_{p_1}, \#_{p_2}, \dots, \#_{p_n}}(E_1)$: Let v_{0_1}, \dots, v_{0_l} be all membership values for a fixed value list v_{p_1}, \dots, v_{p_n} where $(v_{0_i}, v_1, \dots, v_m) \in I^*(E_1)$ holds and the corresponding values are identical: $p_i = j \Rightarrow v_{p_i} = v_j$ for $i = 1, \dots, n$.

$$I^*(\pi_{\#_{p_1}, \#_{p_2}, \dots, \#_{p_n}}(E_1)) = \{(u_0, v_{p_1}, v_{p_2}, \dots, v_{p_n}) | (v_{0_i}, v_1, \dots, v_m) \in I^*(E_1)\} \text{ where}$$

$$u_0 = \begin{cases} I(\vee)(v_{0_1}, \dots, v_{0_l}) & \text{if } l > 1 \\ v_{0_1} & \text{if } l = 1 \end{cases} .$$

Please notice, that duplicate elimination means an OR-aggregation of grouped membership values. For convenience, we apply an n-ary OR-operator since the binary OR-operator as a t-conorm holds commutativity and associativity.

6. *projection* $E = \pi_{\#_{p_1}, \#_{p_2}, \dots, \#_{p_n}}^k(E_1)$: Let v_{0_1}, \dots, v_{0_l} be all membership values for a fixed value list v_{p_1}, \dots, v_{p_n} where $(v_{0_i}, v_1, \dots, v_m) \in I^*(E_1)$ holds, the corresponding values are identical $p_i = j \Rightarrow v_{p_i} = v_j$ for $i = 1, \dots, n$, and

$$u_0 = I(\vee)(v_{0_1}, \dots, v_{0_l}) * \min\left(\frac{v_{0_1} + \dots + v_{0_l}}{k * I(\vee)(v_{0_1}, \dots, v_{0_l})}, 1\right).$$

$$I^*(\pi_{\#_{p_1}, \#_{p_2}, \dots, \#_{p_n}}(E_1)) = \{(u_0, v_{p_1}, v_{p_2}, \dots, v_{p_n}) | (v_{0_i}, v_1, \dots, v_m) \in I^*(E_1)\}$$

7. *selection* $E = \sigma_{y_i \delta y_j}(E_1)$:

$$I^*(\sigma_{y_i \delta y_j}(E_1)) = \{(u_0, v_1, \dots, v_n) | (v_0, v_1, \dots, v_n) \in I^*(E_1) \wedge u_0 = I(\wedge)(v_0, I(\delta)(\hat{y}_i, \hat{y}_j)) \wedge u_0 > 0\}$$

where

$$\hat{y}_i = \begin{cases} v_i & \text{if } y_i \text{ is an attribute} \\ I(y_i) & \text{if } y_i \text{ is a constant name.} \end{cases} \quad \hat{y}_j = \begin{cases} v_j & \text{if } y_j \text{ is an attribute} \\ I(y_j) & \text{if } y_j \text{ is a constant name.} \end{cases}$$

A selection without condition means no restriction: $I^*(\sigma(E_1)) = I^*(E_1)$.

8. *unions*:

- *union* $E = (E_1 \cup E_2)$:

$$I^*((E_1 \cup E_2)) =$$

$$\{(I(\vee)(v_0, w_0), v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \dots, v_k) \in I^*(E_2)\} \cup$$

$$\{(I(\vee)(v_0, 0), v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge \forall w_0. (w_0, v_1, \dots, v_k) \notin I^*(E_2)\} \cup$$

$$\{(I(\vee)(0, w_0), v_1, \dots, v_k) | (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge \forall v_0. (v_0, v_1, \dots, v_k) \notin I^*(E_1)\}.$$
- *weighted union* $E = (E_1 \vec{\cup}_\theta E_2)$:

$$I^*((E_1 \vec{\cup}_\theta E_2)) =$$

$$\{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge u_0 > 0\} \cup$$

$$\{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge \forall w_0. (w_0, v_1, \dots, v_k) \notin I^*(E_2) \wedge u_0 > 0\} \cup$$

$$\{(u_0, v_1, \dots, v_k) | (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge \forall v_0. (v_0, v_1, \dots, v_k) \notin I^*(E_1) \wedge u_0 > 0\}$$
 where

$$u_0 = \begin{cases} I(\vec{\vee}_\theta)(I(\theta), v_0, w_0) & \text{case 1} \\ I(\vec{\vee}_\theta)(I(\theta), v_0, 0) & \text{case 2} \\ I(\vec{\vee}_\theta)(I(\theta), 0, w_0) & \text{case 3} \end{cases}$$

Please notice, that due to the weighted disjunction the weighted union is not associative. The semantics of $E = (E_1 \overleftarrow{\cup}_\theta E_2)$ is analogously defined.

9. intersections:

- *intersection* $E = (E_1 \cap E_2)$:

$$I^*((E_1 \cap E_2)) =$$

$$\{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge$$

$$u_0 = I(\wedge)(v_0, w_0) > 0\}.$$
- *weighted intersection* $E = (E_1 \vec{\cap}_\theta E_2)$:

$$I^*((E_1 \vec{\cap}_\theta E_2)) =$$

$$\{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge u_0 > 0\} \cup$$

$$\{(u_0, v_1, \dots, v_k) | (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge \forall v_0. (v_0, v_1, \dots, v_k) \notin I^*(E_1) \wedge u_0 > 0\}$$
 where

$$u_0 = \begin{cases} I(\vec{\wedge}_\theta)(I(\theta), v_0, w_0) & \text{case 1} \\ I(\vec{\wedge}_\theta)(I(\theta), 0, w_0) & \text{case 2} \end{cases}$$

Please notice, that due to the weighted disjunction the weighted union is not associative. Furthermore notice, that the definition of the weighted intersection does not correspond to a set intersection. The reason is the observation, that a weighted conjunction with one zero-valued operand can produce a non-zero result. The semantics of $E = (E_1 \overleftarrow{\cap}_\theta E_2)$ is analogously defined.

10. combinations:

- *combination* $E = (E_1 \uplus^z E_2)$:

$$I^*((E_1 \uplus^z E_2)) =$$

$$\{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge u_0 > 0\} \cup$$

$$\{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge \forall w_0. (w_0, v_1, \dots, v_k) \notin I^*(E_2) \wedge u_0 > 0\} \cup$$

$$\{(u_0, v_1, \dots, v_k) | (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge \forall v_0. (v_0, v_1, \dots, v_k) \notin I^*(E_1) \wedge u_0 > 0\}$$

where

$$u_0 = \begin{cases} I(\oplus^z)(I(z), v_0, w_0) & \text{case 1} \\ I(\oplus^z)(I(z), v_0, 0) & \text{case 2} \\ I(\oplus^z)(I(z), 0, w_0) & \text{case 3} \end{cases}$$

- *weighted combination* $E = (E_1 \overset{\rightarrow z}{\uplus}_{\theta} E_2)$:

$$\begin{aligned} I^*((E_1 \overset{\rightarrow z}{\uplus}_{\theta} E_2)) = & \\ \{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge u_0 > 0\} \cup & \\ \{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge \forall w_0. (w_0, v_1, \dots, v_k) \notin I^*(E_2) \wedge u_0 > 0\} \cup & \\ \{(u_0, v_1, \dots, v_k) | (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge \forall v_0. (v_0, v_1, \dots, v_k) \notin I^*(E_1) \wedge u_0 > 0\} & \end{aligned}$$

where

$$u_0 = \begin{cases} I(\overset{\rightarrow z}{\oplus}_{\theta})(I(z), I(\theta), v_0, w_0) & \text{case 1} \\ I(\overset{\rightarrow z}{\oplus}_{\theta})(I(z), I(\theta), v_0, 0) & \text{case 2} \\ I(\overset{\rightarrow z}{\oplus}_{\theta})(I(z), I(\theta), 0, w_0) & \text{case 3} \end{cases}$$

The semantics of $E = (E_1 \overset{\leftarrow z}{\uplus}_{\theta} E_2)$ is analogously defined.

11. *difference* $E = (E_1 \setminus E_2)$:

$$\begin{aligned} I^*((E_1 \setminus E_2)) = & \\ \{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge \forall w_0. (w_0, v_1, \dots, v_k) \notin I^*(E_2) \wedge u_0 > 0\} \cup & \\ \{(u_0, v_1, \dots, v_k) | (v_0, v_1, \dots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \dots, v_k) \in I^*(E_2) \wedge u_0 > 0\} & \text{ where} \\ u_0 = \begin{cases} I(\wedge)(v_0, I(\neg)(0)) & \text{case 1} \\ I(\wedge)(v_0, I(\neg)(w_0)) & \text{case 2} \end{cases} & \end{aligned}$$

12. *cartesian product* $E = E_a \times E_b$:

$$\begin{aligned} I^*(E_a \times E_b) = & \\ \{(u_0, v_1, \dots, v_k, w_1, \dots, w_l) | (v_0, v_1, \dots, v_k) \in I^*(E_a) \wedge (w_0, w_1, \dots, w_l) \in I^*(E_b) \wedge & \\ u_0 = I(\wedge)(v_0, w_0)\} & \end{aligned}$$

13. *join* $E = E_a \bowtie_{\#a_1=\#b_1, \dots, \#a_n=\#b_n} E_b$:

$$\begin{aligned} I^*(E_a \bowtie_{\#a_1=\#b_1, \dots, \#a_n=\#b_n} E_b) = & \\ \{(u_0, v_1, \dots, v_k, w_1, \dots, w_l) | (v_0, v_1, \dots, v_k) \in I^*(E_a) \wedge (w_0, w_1, \dots, w_l) \in I^*(E_b) \wedge & \\ i \in \{1, \dots, n\}. v_{a_i} = w_{b_i} \wedge u_0 = I(\wedge)(v_0, w_0)\} & \end{aligned}$$

A join with an empty condition produces the cartesian product.

14. *cut* $E = \overset{\theta}{\succ} (E_1)$:

$$I^*(\overset{\theta}{\succ} (E_1)) = \left\{ (u_0, v_1, \dots, v_m) | (v_0, v_1, \dots, v_m) \in I^*(E_1) \wedge u_0 = \begin{cases} 1 & \text{if } v_0 \geq I(\theta) \\ 0 & \text{otherwise} \end{cases} \right\}$$

15. *softcut* $E = \overset{\theta}{\succ} (E_1)$:

$$I^*(\overset{\theta}{\succ} (E_1)) = \left\{ (u_0, v_1, \dots, v_m) | (v_0, v_1, \dots, v_m) \in I^*(E_1) \wedge u_0 = \begin{cases} v_0 & \text{if } v_0 \geq I(\theta) \\ 0 & \text{otherwise} \end{cases} \right\}$$

Please note, that every operation, except for *cut* and *softcut*, yields tuples with nonzero membership values.

6 Reducing SDC to SA

In this section we show that the similarity algebra SA is as expressive as the similarity domain calculus SDC .

Theorem 6.0.1 *Let $SDC = (U, \mathbf{X}, \Delta, \mathbf{C}, \mathbf{D}, Dom, \mathbf{R}, \Theta, \mathbf{Z})$ be a similarity relational domain calculus and $SA = (U, \Delta, \mathbf{C}, \mathbf{D}, Dom, \mathbf{R}, \Theta, \mathbf{Z})$ be a similarity relational algebra. For any SDC formula F there is an SA expression E that their queries are equivalent for any corresponding semantics:*

$$\{V(X_1), \dots, V(X_n) | I_V^*(F(X_1, \dots, X_n)) > 0\} = \{(v_1, \dots, v_n) | (v_0, v_1, \dots, v_n) \in I^*(E) \wedge v_0 > 0\}.$$

Proof We prove Theorem 6.0.1 by constructively defining a *mapping* φ of an SDC formula F to a similarity algebra expression $E = \varphi(F)$:

1. case $F = R(Y_1, Y_2, \dots, Y_m)$:

$$\varphi(F) = \pi_{\#_{v_1}, \dots, \#_{v_n}}(\sigma_{cond_c, cond_v}(R))$$

where

- $cond_c := \#_{i_1} = C_{i_1}, \dots, \#_{i_l} = C_{i_l}$ where Y_{i_1}, \dots, Y_{i_l} are all constant names C_{i_1}, \dots, C_{i_l} ,
- $cond_v := \#_{j_1} = \#_{k_1}, \dots, \#_{j_o} = \#_{k_o}$ where $Y_{j_1} = Y_{k_1}, \dots, Y_{j_o} = Y_{k_o}$ are all pairs of equally named variables reduced by reflexivity, symmetry, and transitivity on the attribute positions, and
- $\#_{v_1}, \dots, \#_{v_n}$ lists uniquely the attributes for all variables Y_{v_i} in the order of the variables.

2. case $F = Y_1 \delta Y_2$:

- if $Y_1, Y_2 \in \mathbf{X} \wedge Y_1 \neq Y_2$ then $\varphi(F) = \sigma_{\#_1 \delta \#_2}(Dom_D \times Dom_D)$;
- if $Y_1, Y_2 \in \mathbf{X} \wedge Y_1 = Y_2$ then $\varphi(F) = \pi_{\#_1}(\sigma_{\#_1 \delta \#_2}(\pi_{\#_1, \#_2}(Dom_D)))$;
- if $Y_1 \in \mathbf{X}$ and Y_2 is a constant $C \in \mathbf{C}$ then $\varphi(F) = \sigma_{\#_1 \delta C}(Dom_D)$;
- if $Y_2 \in \mathbf{X}$ and Y_1 is a constant $C \in \mathbf{C}$ then $\varphi(F) = \sigma_{C \delta \#_1}(Dom_D)$;
- if Y_1 and Y_2 are constants $C_1, C_2 \in \mathbf{C}$, respectively, then $\varphi(F) = \sigma_{C_1 \delta C_2}(\mathbf{1})$;

where $D = Dom(\delta)$.

3. case $F = (F_a(X_{a1}, \dots, X_{ak}) \wedge F_b(X_{b1}, \dots, X_{bl}))$:

$$\varphi(F) = \pi_{\#_{p_1}, \dots, \#_{p_n}}(\varphi(F_a) \bowtie_{cond_c} \varphi(F_b))$$

where

- $cond_c := \#_{v_1} = \#_{w_1}, \dots, \#_{v_m} = \#_{w_m}$ where $X_{av_i} = X_{bw_i} \forall i = 1, \dots, m$ and
- $\#_{p_1}, \dots, \#_{p_n}$ lists uniquely the corresponding attributes for all variables $X_{a1}, \dots, X_{ak}, X_{b1}, \dots, X_{bl}$ in the order of the variables.

4. weighted **and**-cases: case $F = (F_a(X_{a_1}, \dots, X_{a_k}) \overset{\rightarrow}{\wedge}_\theta F_b(X_{b_1}, \dots, X_{b_l})) :$

$$\varphi(F) = E_a \overset{\rightarrow}{\cap}_\theta E_b$$

where

$$E_a = \pi_{\#_{i_1}, \dots, \#_{i_m}} (\varphi(F_a) \times \text{dom}(X_{b_{o_1}}) \times \dots \times \text{dom}(X_{b_{o_{z-k}}}))$$

$$E_b = \pi_{\#_{j_1}, \dots, \#_{j_m}} (\varphi(F_b) \times \text{dom}(X_{a_{p_1}}) \times \dots \times \text{dom}(X_{a_{p_{z-l}}}))$$

and the following conditions hold:

- $\{X_{a_1}, \dots, X_{a_k}, X_{b_1}, \dots, X_{b_l}\}$ correspond bijectively to $\text{att}(E_a)$ and $\text{att}(E_b)$, respectively,
- the attributes of E_a and of E_b occur in the order of the variables, and
- $z = |\{X_{a_1}, \dots, X_{a_k}, X_{b_1}, \dots, X_{b_l}\}|$.

The case for the left-oriented conjunction $\overset{\leftarrow}{\wedge}_\theta$ is defined analogously.

5. **or**-cases (unweighted and weighted): These cases are defined analogously to the weighted **and**-cases. \vee is mapped to \cup , $\overset{\leftarrow}{\vee}_\theta$ is mapped to $\overset{\leftarrow}{\cup}_\theta$, and $\overset{\rightarrow}{\vee}_\theta$ is mapped to $\overset{\rightarrow}{\cup}_\theta$.

6. **universal operators**-cases (unweighted and weighted): These cases are defined analogously to the weighted **and**-cases. \oplus^z is mapped to \uplus^z , $\overset{\leftarrow}{\oplus}_\theta$ is mapped to $\overset{\leftarrow}{\uplus}_\theta$, and $\overset{\rightarrow}{\oplus}_\theta$ is mapped to $\overset{\rightarrow}{\uplus}_\theta$.

7. case $F = (\neg F_1(X_1, \dots, X_m)) :$

$$\varphi(F) = (\text{Dom}(X_1) \times \dots \times \text{Dom}(X_m)) \setminus \varphi(F_1).$$

If F_1 has no variables ($m = 0$) then $\varphi(F) = \mathbf{1} \setminus \varphi(F_1)$.

8. case $F = (\exists X F_1(X_1, \dots, X_{l-1}, X, X_{l+1}, \dots, X_m)) :$

$$\varphi(F) = \pi_{\#_{p_1}, \dots, \#_{p_{m-1}}} (\varphi(F_1(X_1, \dots, X_{l-1}, X, X_{l+1}, \dots, X_m)))$$

where $\#_{p_1}, \dots, \#_{p_{m-1}}$ lists the corresponding attributes for all variables $X_1, \dots, X_{l-1}, X_{l+1}, \dots, X_m$ in the order of the variables.

9. case $F = (\exists_k X F_1(X_1, \dots, X_{l-1}, X, X_{l+1}, \dots, X_m)) :$

$$\varphi(F) = \pi_{\#_{p_1}, \dots, \#_{p_{m-1}}}^k (\varphi(F_1(X_1, \dots, X_{l-1}, X, X_{l+1}, \dots, X_m)))$$

where $\#_{p_1}, \dots, \#_{p_{m-1}}$ lists the corresponding attributes for all variables $X_1, \dots, X_{l-1}, X_{l+1}, \dots, X_m$ in the order of the variables.

case $F = (\forall X F_1) : \varphi(F) = \varphi((\neg(\exists X(\neg F_1))))$.

10. case $F = (\forall_k X F_1) : \varphi(F) = \varphi((\neg(\exists_k X(\neg F_1))))$.

11. case $F = (\overset{\theta}{\succ} F_1) : \varphi(F) = \overset{\theta}{\succ} (\varphi(F_1))$.

12. case $F = (\overset{\theta}{\neq} F_1) : \varphi(F) = \overset{\theta}{\neq} (\varphi(F_1))$.

□

Mapping an SDC-formula to the similarity algebra can produce similarity algebra expressions containing the Dom-operation. In such cases, the expression is domain dependent. If, however, an SDC-formula is evaluable and we map it to the similarity algebra then we can replace all occurring Dom-operations by similarity algebra expressions over database relations and obtain domain independent algebra expressions.

Theorem 6.0.2 *Let SDC = (U, X, Δ, C, D, Dom, R, Θ, Z) be a similarity relational domain calculus and SA = (U, Δ, C, D, Dom, R, Θ, Z) be a similarity relational algebra. For any evaluable SDC formula F there is a domain independent SA expression E that their queries are equivalent for any corresponding semantics:*

$$\{V(X_1), \dots, V(X_n) | I_V^*(F(X_1, \dots, X_n)) > 0\} = \{(v_1, \dots, v_n) | (v_0, v_1, \dots, v_n) \in I^*(E) \wedge v_0 > 0\}.$$

Proof We prove Theorem 6.0.2 by modifying the mapping φ from the proof of theorem 6.0.1 to the mapping φ^* of an evaluable SDC formula F to a similarity algebra expression $\varphi^*(F)$.

Let $E = \{X_1, X_2, \dots, X_n | F(X_1, X_2, \dots, X_n)\}$ be an evaluable SDC query and let

$$rel(X) = ((E_1) \cap \dots \cap (E_m))$$

be a function on variables involved (free or bound) in F to obtain corresponding database values. The SA expressions E_i with $i = 1, \dots, m$ corresponding to X are derived from the *gen* relation $\{E_i | gen(X, F, E_i)\}$ if X is free in F . Otherwise, if X is existentially bound in F by the subformula $\exists X F_1$ then E_i with $i = 1, \dots, m$ are derived from the *con* relation $\{E_i | con(X, F_1, E_i)\}$. For a variable X universally bound in F by the subformula $\forall X F_1$ the E_i expressions with $i = 1, \dots, m$ are derived from the *con* relation $\{E_i | con(X, \neg F_1, E_i)\}$. Since the symbol \perp from a *con* relation stands for an empty relation, it is removed from any disjunction within $rel(X)$.

1. For every atom $Y_1 \delta Y_2$ within F replace the corresponding algebra term

- a) $\sigma_{\#_1 \delta \#_2}(Dom_D \times Dom_D)$ in $\varphi(F)$ by $\sigma_{\#_1 \delta \#_2}(rel(Y_1) \times rel(Y_2))$ if $Y_1, Y_2 \in \mathbf{X} \wedge Y_1 \neq Y_2$,
- b) $\pi_{\#_1}(\sigma_{\#_1 \delta \#_2}(\pi_{\#_1, \#_2}(Dom_D)))$ in $\varphi(F)$ by $\pi_{\#_1}(\sigma_{\#_1 \delta \#_2}(\pi_{\#_1, \#_2}(rel(Y_1))))$ if $Y_1, Y_2 \in \mathbf{X} \wedge Y_1 = Y_2$,
- c) $\sigma_{\#_1 \delta C}(Dom_D)$ in $\varphi(F)$ by $\sigma_{\#_1 \delta Y_2}(rel(Y_1))$ if $Y_1 \in \mathbf{X}, Y_2 \in \mathbf{C}$, and
- d) $\sigma_{C \delta \#_1}(Dom_D)$ in $\varphi(F)$ by $\sigma_{Y_1 \delta \#_1}(rel(Y_2))$ if $Y_1 \in \mathbf{C}, Y_2 \in \mathbf{X}$.

2. weighted conjunctions, disjunctions, and combinations: For every weighted conjunction

$$(F_a(X_{a_1}, \dots, X_{a_k}) \overset{\theta}{\wedge} F_b(X_{b_1}, \dots, X_{b_l}))$$

in F replace the corresponding algebra term $E_a \overset{\theta}{\wedge} E_b$ in $\varphi(F)$ by $E'_a \overset{\theta}{\wedge} E'_b$

where

$$E'_a = (\pi_{\#_{i_1}, \dots, \#_{i_m}}(\varphi(F_a) \times rel'(X_{b_{o_1}}) \times \dots \times rel'(X_{b_{o_{z-k}}}))$$

$$E'_b = (\pi_{\#_{j_1}, \dots, \#_{j_m}}(\varphi(F_b) \times rel'(X_{a_{p_1}}) \times \dots \times rel'(X_{a_{p_{z-l}}}))$$

and the following conditions hold:

- $\{X_{a1}, \dots, X_{ak}, X_{b1}, \dots, X_{bl}\}$ correspond bijectively to $att(E'_a)$ and $att(E'_b)$, respectively,
- the attributes of E'_a and of E'_b occur in the same order, and
- $z = |\{X_{a1}, \dots, X_{ak}, X_{b1}, \dots, X_{bl}\}|$.
-

$$rel'(X) = \begin{cases} rel(X) \cup \mathbf{0} & \text{if } X \text{ is bound to } \exists \text{ or } \forall \\ rel(X) & \text{otherwise} \end{cases}$$

These cases for $\overleftarrow{\wedge}_\theta, \overrightarrow{\vee}, \overleftarrow{\vee}_\theta, \overrightarrow{\vee}_\theta, \oplus^z, \oplus_\theta^z$, and $\overrightarrow{\oplus}_\theta^z$ are defined analogously.

3. For every negation $(\neg F_1(X_1, \dots, X_m))$ within F replace the corresponding algebra term $(Dom(X_1) \times \dots \times Dom(X_m)) \setminus \varphi(F_1)$ in $\varphi(F)$ by $(rel(X_1) \times \dots \times rel(X_m)) \setminus \varphi(F_1)$.

□

Now we are able to formulate weighted similarity queries in declarative manner using *SDC* and to reduce them to an equivalent algebra expression. Thus, the evaluation of those similarity queries can be processed based on the proposed *SA*.

7 The Phantom Problem

Assume the following evaluable calculus formula $\exists X(R_1(X) \vee R_2(C_1))$ is given and the relation $I(R_1)$ is an empty relation. If additionally the second relation contains the constant value, then the formula is satisfied and returns true.

Consider now the mapping to the similarity algebra. Following the mapping rule for the \vee -construct we have to unite (\cup) two algebra expressions. Since both expressions are not union compatible the right expression needs to be combined with the relation $I(R_1)$ applying the cartesian product. However, in our case the relation $I(R_1)$ is empty and the cartesian product with an empty relation produces always an empty relation. Therefore, the complete algebra expression returns no tuple. This contradicts the result of the calculus expression and we call it the *phantom problem*.

We solve this problem by never using an empty relation for becoming union compatible. This is achieved by applying the ' $\cup \mathbf{0}$ ' operation to R_1 ¹².

The phantom problem becomes worse if we replace $\exists X$ by $\exists_k X$: $(\exists_k X(R_1(X) \vee R_2(C_1)))$. In this case, the formula is not domain independent anymore. The reason is that the result depends on the number of elements of the domain for X . Therefore, for evaluable formulas we do not allow any disjunction or weighted conjunction with a subformula independent from X below an $\exists_k X$ or a $\forall_k X$ construct¹³.

8 Example

In order to demonstrate the potential of our approach we will demonstrate the transformation process on an example using a fabric seller database. Information of different fabrics is stored

¹²See rule 2 in the proof for Theorem 6.0.2.

¹³This test is performed in rules 4 and 5 of the Definition 4.1 by checking the ' \perp ' symbol.

in Table *Fabric*. To each fabric there exist several images that are stored in Table *Image*. A small fragment of these tables is shown below. For similarity calculation between the given images and those in the database different similarity operators are available, e.g. \sim_C to determine the similarity regarding the color feature.

Fabric		
FId	Name	Quality
F001	tartan_0815	high
F002	tartan_0816	low
F003	blue_stripes	high
...

Image		
IId	FId	Image
I001	F001	tartan_0815.1.jpg
I002	F001	tartan_0815.2.jpg
I003	F002	tartan_0816.1.jpg
I004	F003	blue_stripes.1.jpg
...

Consider the following query:

»Retrieve name and quality of those fabrics that have a high quality or that match the given query image (C_{image}) in color and texture. Thereby, color is twice as important as texture and the quality criteria is three times less important than the similarity to the given image.«

Formulating queries in an declarative way is much more comfortable to the user since the user only describes what he is interested in and not how it can be obtain from the database. Therefore, we can directly formulate this query as *SDC*-expression, which is illustrated in Fig. 1. Compared to the algebra expression in Fig. 2 the calculus expression is less complex and easier to understand. Of course, querying directly in calculus is still too difficult for

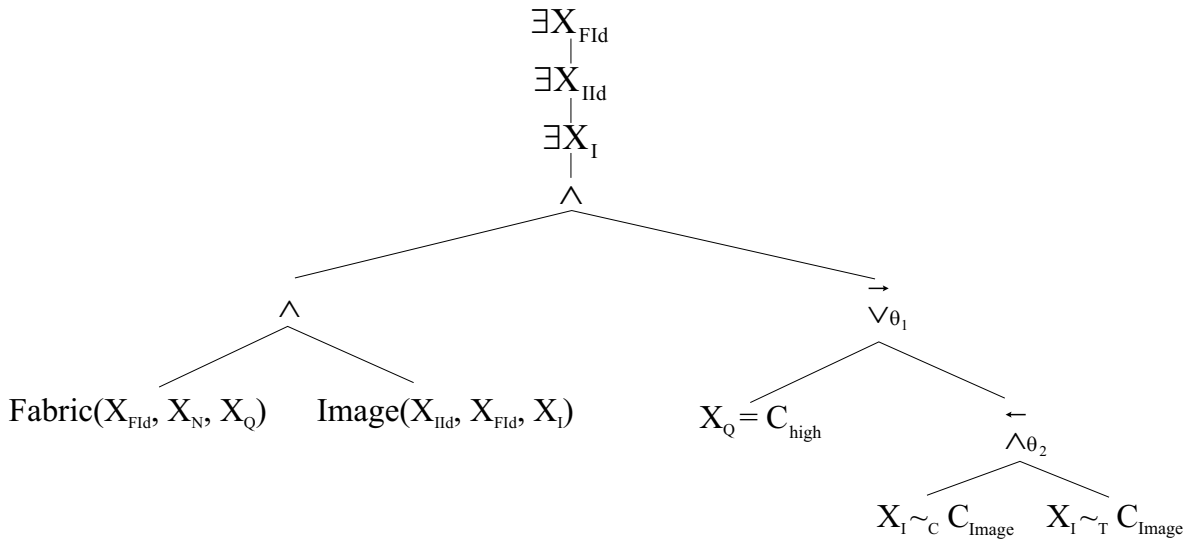


Figure 1: Query tree of the *SDC*-expression with the free variables X_N and X_Q .

a non-expert user. For that reason we are currently designing a graphical QBE-like query language. From there, a mapping onto our calculus can be easily performed.

Considering our query in *SDC* we first need to determine if the query is evaluable or not. Since $gen()$ holds for the free variables X_N and X_Q , e.g. $gen(X_N, F, \pi_{\#2}(Fabric))$ is true,

and $con()$ holds for the bound variables X_{FId} , X_{IId} , and X_I the query is evaluable. Mapping the SDC -expression yields a complex \mathcal{SA} -expression, which corresponds to the query tree shown in Figure 2.

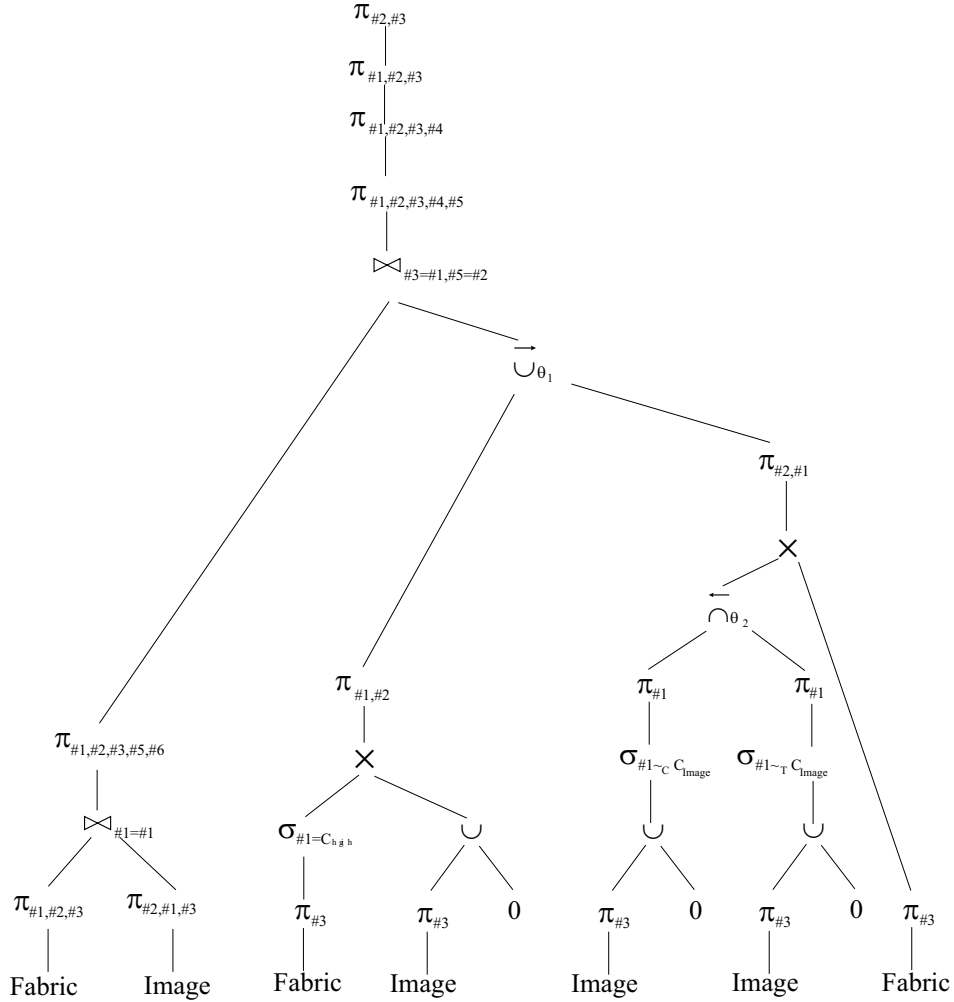


Figure 2: Query tree of the generated \mathcal{SA} -expression.

We use Fagin’s weighting formula [12] for the weighted combination of similarity values. As underlying scoring function we employ the functions min for conjunction and max for disjunction. In order to work appropriate with the weighting formula we transform our operator weights to operand weights as shown in Section 3.

Due to the specified user preferences, which state that the quality criteria is three times less important than the similarity criteria we obtain for $\overrightarrow{\vee}_{\theta_1}$ the weight $\theta_1 = 1/2$. The corresponding operand weights are therefore $\theta_{quality} = 1/4$ and $\theta_{\sim} = 3/4$. Since color should be twice as important as texture, the weight θ_2 for the operation $\overleftarrow{\wedge}_{\theta_2}$ is $1/3$, which corresponds to the operand weights $\theta_{\sim_C} = 2/3$ and $\theta_{\sim_T} = 1/3$. The constant C_{high} is evaluated to the string ‘high’ and the constant C_{Image} corresponds to a query image. The calculated similarity values for the subformulas are summarized in Table *Similarity Values*. The query result is

shown in Table *Result*. The final projection eliminates duplicates. Thus, the tuple with the highest score of the duplicates is taken. For fabric *tartan_0815* the tuple with score 0.9 is chosen, whereas the duplicate tuple with score 0.69 is omitted.

IId	\sim_C	\sim_T	$\overleftarrow{\wedge}_{\theta_2}$	$\overrightarrow{\vee}_{\theta_1}$
I001	0.7	0.2	0.37	0.69
I002	0.8	0.9	0.80	0.90
I003	0.9	0.4	0.57	0.57
I004	0.1	0.8	0.10	0.55
...

Name	Quality	Score
tartan_0815	high	0.90
tartan_0816	low	0.57
blue_stripes	high	0.55
...

9 Conclusion and Future Work

Our approach defines a framework for transforming declarative similarity queries into an appropriate expression of an extended relational algebra. We enhanced traditional relational domain calculus by vagueness and the aspect of weighting. We left the specification of exact fuzzy-operations and weighted scoring functions unspecified, so that our approach can be adapted and flexible extended to meet the needs of various scenarios, e. g. image retrieval or other multimedia applications.

So far, our proposed language comprises core functionality only. We provide adequate operators for dealing with imprecision and weights. Further, we proposed the parameterisable quantifiers \exists_k and \forall_k . In future, we will expand our language by new constructs. We plan to add a similarity join [2, 1] as well as *top*- and *cut*-operations [6]. In addition, we intend to support aggregations and other functions on result variables. Another aspect we want to consider is to weight the operators itself, that is, to modify their behavior by a parameter, e. g. to soften a conjunction in direction to the disjunction. Since formulating queries in a calculus is not very user friendly, we aim to design a graphical query language in a QBE-like fashion. Then, a mapping onto our similarity calculus can be easily performed.

Reducing calculus expressions to algebra produces often very complex expressions requiring subsequent algebraic optimization. We are planning to develop appropriate optimization strategies. Therefore, special optimization rules adapted to our mapping rules need to be developed. In addition to a subsequent optimization we are working on implicit optimization during the mapping. Furthermore, we aim to extend the relational database model in order to explicitly store and query imprecise data.

References

- [1] S. Adali, B. Bonatti, M. L. Sapino, and V. S. Subrahmanian. A Multi-Similarity Algebra. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 402–413, Seattle, Washington, USA, 1998.
- [2] S. Atnafu, L. Brunie, and H. Kosch. Similarity-Based Operators and Query Optimization for Multimedia Database Systems. In Michel E. Adiba, Christine Collet, and Bipin C. Desai, editors, *International Database Engineering & Applications Symposium, IDEAS '01, July 16-18, 2001, Grenoble, France, Proceedings*, pages 346–355. IEEE Computer Society, 2001.

-
- [3] J. Biskup. *Grundlagen von Informationssystemen*. Vieweg-Verlag, Braunschweig, Wiesbaden, 1995.
- [4] N. Bolloju. A Calculus for Fuzzy Queries on Fuzzy Entity-Relationship Model. Technical Report 94/26, Department of Information Systems at the City Polytechnic of Hong Kong, 1994.
- [5] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Region-based image querying. In *Proc. of the IEEE Workshop CVPR '97 Workshop on Content-Based Access of Image and Video Libraries, Puerto Rico*, pages 42–49, 1997.
- [6] P. Ciaccia, D. Montesi, W. Penzo, and A. Trombetta. Imprecision and user preferences in multimedia queries: A generic algebraic approach. In K.-D. Schewe and B. Thalheim, editors, *FoIKS: Foundations of Information and Knowledge Systems, First International Symposium, FoIKS 2000, Burg, Germany, February 14-17, 2000*, volume 1762 of *Lecture Notes in Computer Science*, pages 50–71. Springer, 2000.
- [7] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, June 1970.
- [8] E. F. Codd. A Database Sublanguage Founded on the Relational Calculus. In *ACM SIGFIDET Workshop on Data Description, Access and Control*, pages 35–61, nov 1971.
- [9] E. F. Codd. Relational Completeness of Data Base Sublanguages. In R. Rustin, editor, *Data Base Systems*, volume 6, pages 65–98. Prentice Hall, Englewood Cliffs, NJ, 1972.
- [10] E. F. Codd. Relational Database: A Practical Foundation for Productivity. *Communications of the ACM*, 25(2):109–117, February 1982.
- [11] C. J. Date. A Note on the Relational Calculus. *ACM SIGMOD Record*, 18(4):12–16, December 1989.
- [12] R. Fagin and E. L. Wimmers. A Formula for Incorporating Weights into Scoring Rules. *Special Issue of Theoretical Computer Science*, 2000.
- [13] N. Fuhr and T. Rilleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Databases Systems. *ACM Transactions on Information Systems (TOIS)*, 15(1):32–66, January 1997.
- [14] J. Galindo, J. M. Medina, O. Pons, and J. C. Cubero. A Server for Fuzzy SQL Queries. In T. Andreasen, H. Christiansen, and H. L. Larsen, editors, *Flexible Query Answering Systems, Third International Conference, FQAS'98, Roskilde, Denmark, May 13-15, 1998*, volume 1495 of *Lecture Notes in Computer Science*, pages 164–174. Springer, 1998.
- [15] A. V. Gelder and R. W. Topor. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems*, 16(2):235–278, 1991.
- [16] W. Kieling. Foundations of preferences in database systems. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB), Hong Kong, China*, pages 311–322, 2002.

- [17] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, MD, 1983.
- [18] M. Ortega, Y. Rui, K. Chakrabarti, K. Porkaew, S. Mehrotra, and T. S. Huang. Supporting Ranked Boolean Similarity Queries in MARS. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):905–925, 1998.
- [19] N. Schulz and I. Schmitt. A Survey of Weighted Scoring Rules in Multimedia Database Systems. Preprint 7, Fakultät für Informatik, Universität Magdeburg, 2002.
- [20] N. Schulz and I. Schmitt. Relevanzwichtung in komplexen hnlichkeitsanfragen. In G. Weikum, H. Schöning, and E. Rahm, editors, *Datenbanksysteme in Business, Technologie und Web, BTW'03, 10. GI-Fachtagung, Leipzig, Februar 2003*, Lecture Notes in Informatics (LNI) Volume P-26, pages 187–196, Bonn, 2003. Gesellschaft für Informatik.
- [21] S.Y. Sung. A Linear Transform Scheme for Combining Weights into Scores. *Technical Report, Rice University*, 1998.
- [22] Y. Takahashi. Fuzzy Database Query Languages and Their Relational Completeness Theorem. *IEEE Transaction on Knowledge and Data Engineering*, 5(1):122–125, February 1993.
- [23] J. D. Ullman. *Principles of Database Systems*. Computer Science Press, 1982.
- [24] W. G. Waller and D. H. Kraft. A mathematical model for a weighted boolean retrieval system. *Information Processing and Management*, 15(5):235–245, 1979.
- [25] Lofti A. Zadeh. Fuzzy Logic. *IEEE Computer*, 21(4):83–93, April 1988.
- [26] H.-J. Zimmermann and P. Zysno. Latent connectives in human decision making. *Fuzzy Sets and Systems*, 4:37–51, 1980.