A Hierarchically Typed Relation Algebra

Patrick Roocks

Institut für Informatik, Universität Augsburg, D-86135 Augsburg, Germany
roocks@informatik.uni-augsburg.de

Abstract. We consider a typed relation algebra which is used for a calculus of database preferences. So far, the typing mechanism does not cover unions of differently typed elements, which means that the global identity and the greatest element are not typable in the calculus. We suggest a hierarchical type structure to remedy this.

1 Introduction

Database preferences can be modelled as homogeneous binary relations over the type domains given by their attributes (i.e., database columns). In our previous papers [MRE12, MR12] we developed a calculus where an algebra of typed elements is used to represent database preferences. But the union of differently typed elements is up to now not typable with our mechanism. This implies that the global 1 and $\top$ elements, algebraically equivalent to the sum of all typed 1 and $\top$ elements, are not typable. We suggest a hierarchy of types, which covers such unions and show some properties.

A similar concept of a type hierarchy in the scope of databases is described in the *Higher Order Entity Relationship Model (HERM)* [Tha93], where "clusters" form the counterpart to our multitypes.

2 Typed tuples and relations

In this section we recapitulate the definions of typed tuples and the join from [MRE12]. First we define a typing mechanism for tuples and a join operation for sets of typed tuples.

2.1 Typed tuples

Definition 2.1. Let $\mathcal{A}$ be a set of *attribute names*. For each $A \in \mathcal{A}$ we assume a set D_A, called the *type domain* of A. We define the following notions:

- A *type* is a subset $T \subseteq \mathcal{A}$ with $T \neq \emptyset$.
- A *T-tuple* is a mapping

$$t : T \to \bigcup_{A \in \mathcal{A}} D_A \ \text{ where } \ \forall\, A \in T : t(A) \in D_A.$$

– The type domain D_T is the set of all T-tuples, i.e., $D_T = \prod_{A \in T} D_A$.
– The set $\mathcal{U} =_{df} \bigcup_{T \subseteq \mathcal{A}} D_T$ is called the *universe*.
– For a tuple t and a set of tuples M we introduce the following abbreviations:

$$t :: T \Leftrightarrow_{df} t \in D_T, \qquad M :: T \Leftrightarrow_{df} M \subseteq D_T.$$

Definition 2.2 (Join). The *join* of two types T_1, T_2 is the union of their attributes: $T_1 \bowtie T_2 =_{df} T_1 \cup T_2$. For sets of tuples $M_i :: T_i$ $(i = 1, 2)$, the join is defined as the set of all consistent combinations of M_i-tuples:

$$M_1 \bowtie M_2 =_{df} \{t :: T_1 \bowtie T_2 \mid t|_{T_i} \in M_i, i = 1, 2\} .$$

The term "join" is used to point out the analogy to the *natural join* in databases. When two sets are joined, only those tuples are joined who have identical values in the common attributes of both sets. We illustrate this in the following example:

Example 2.3. Assume a database of cars with a unique ID and further attributes for model and horsepower. Hence the attribute names, i.e., the types, are ID, model and hp. The tuples are written as explicit mappings. Assume the following sets:

$$M_1 =_{df} \{\{\text{ID} \mapsto 1, \text{ model} \mapsto \text{'BMW 7'}\}, \{\text{ID} \mapsto 3, \text{ model} \mapsto \text{'Mercedes CLS'}\}\},$$
$$M_2 =_{df} \{\{\text{ID} \mapsto 2, \text{ hp} \mapsto 230\}, \{\text{ID} \mapsto 3, \text{ hp} \mapsto 315\}\}.$$

The sets have the types $M_1 :: \text{ID} \bowtie \text{model}$ and $M_2 :: \text{ID} \bowtie \text{hp}$. Now we consider the join $M_1 \bowtie M_2 :: \text{ID} \bowtie \text{model} \bowtie \text{hp}$. We have $(\text{ID} \bowtie \text{model}) \cap (\text{ID} \bowtie \text{hp}) = \text{ID}$. The only tuple $t :: \text{ID} \bowtie \text{model} \bowtie \text{hp}$ which fulfills both $t|_{T_1} \in M_1$ and $t|_{T_2} \in M_2$ is the one with $\text{ID} \mapsto 3$. Hence the join is given by:

$$M_1 \bowtie M_2 = \{\{\text{ID} \mapsto 3, \text{ model} \mapsto \text{'Mercedes CLS'}, \text{ hp} \mapsto 315\}\} .$$

In [MRE12] also some useful properties of the $\bowtie$-operator are given, e.g., $\bowtie$ is associative, commutative, distributes over $\cup$ and is isotone.

In [MRE12, MR12] this concept is applied to database preferences, which are modelled as homogeneous binary relations. To this end the typing mechanism above is extended to relations. This is straight forward by the definitions

$$(t_1, t_2) :: T^2 \;\Leftrightarrow_{df}\; t_i \in D_T, \qquad R :: T^2 \;\Leftrightarrow_{df}\; R \subseteq D_T \times D_T.$$

Analogously to the case of sets of tuples, a join operator on relations can be defined, and similar properties hold. But we will not go into detail here, as it is not important for the rest of the paper.

2.2 Typed abstract relation algebra

Now we will generalize the above definitions. Homogeneous binary relations over a set form an idempotent semiring with choice $\cup$ and composition ";", which have $\emptyset$ and the identity relation as their respective units. In the abstract relation algebra choice is denoted by "+" and composition by ";". We assume that for every type T our algebra contains a test $1_T \leq 1$ representing the type domain D_T. Sets $M \subseteq D_T$ are modelled as *tests* p, and can be characterised as sub-identities, i.e., $p \leq 1_T$. Thereby "$\leq$" is the *subsumption order*, which is defined by $x \leq y \Leftrightarrow x + y = y$.

For general elements, by convention named $a, b, c, ...$, we define the type assertions:

$$a :: T^2 \quad \Leftrightarrow_{df} \quad a \leq 1_T \cdot a \cdot 1_T \ .$$

The tests, i.e. sub-identities, are typed as follows:

$$p :: T \quad \Leftrightarrow_{df} \quad p \leq 1_T \ .$$

For types $T \neq U$ we assume $1_T \cdot 1_U = 0$, where 0 is the smallest element in the algebra and corresponds to the empty set in the concrete instance. By fulfilling all type assertions, it has no explicit type. Further we assume a largest element $\top_T$ corresponding to the full relation. For every $x :: T^2$ we have $0 \leq x \leq \top_T$.

Tests of arbitrary type fulfil $p \leq 1$, where 1 represents the universe $\mathcal{U}$. And we assume a greatest element $\top$ with $x \leq \top$ for all x, independent of their type, if any. Note that 1 and $\top$ have up to now no type.

3 Multitypes

Assume distinct attributes A, B and consider two tests $p_A :: \{A\}$ and $p_B :: \{B\}$. The supremum of two elements (i.e., the union in the relational setting) can be denoted as $p_A + p_B$; but can we also determine its type in our calculus? Note that $p_A + p_B :: \{A, B\}$ does not hold: By definition we have $\{A\} \cup \{B\} = \{A\} \bowtie \{B\}$, but $p_A + p_B$ does not contain the join of elements of types $\{A\}$ and $\{B\}$. In fact, elements like $a + b$ where a and b are differently typed, are not typable with our "::" operator.

For the preference-related operations occurring in [MRE12, MR12] this does not matter as there such elements do not occur. But for the uniformity of the algebra this is a drawback. Note that for the subsumption order the following is valid by definition:

$$p_A \ \leq \ p_A + p_B \ \leq \ 1 \ \leq \ \top$$

But $(p_A + p_B)$, 1 and $\top$ are all not typable with the "::" operator. The reason for this is that "inhomogeneous unions" are not covered by our current typing mechanism. While this seems to be a minor technical flaw for the suprema like $p_A + p_B$ it is a lack of uniformity that the "global" identity and top element is untyped. This problem asks for a general and formally consistent solution.

3.1 A type hierarchy

Consider again the definition of the universe $\mathcal{U} = \bigcup_{T \subseteq \mathcal{A}} D_T$ (which is represented by "1"). There T ranges over all subsets of the attribute names $\mathcal{A}$, except the empty set ($D_\emptyset$ is not defined). Hence the set $T_\mathcal{U} =_{df} \mathcal{P}(\mathcal{A}) - \emptyset$, where $\mathcal{P}$ denotes the power set, gives us the types of all type domains contained in $\mathcal{U}$. Thus it seems to be reasonable to say that $T_\mathcal{U}$ is the type of $\mathcal{U}$. We formalize this idea in the following, by suggesting a *type hierarchy.*

Definition 3.1. For a finite set of attribute names $\mathcal{A}$ we define:

1. The set of fundamental types $\mathcal{F}$, consisting of:
   (a) *Base types*: If $A \in \mathcal{A}$, then $\{A\}$ is a base type.
   (b) *Complex types*: Let T_1 and T_2 both fundamental (i.e., base or complex) types. Then $T_1 \cup T_2$ is also a complex type.
   In summary, the set of fundamental types equals $\mathcal{F} = \mathcal{P}(\mathcal{A}) - \emptyset$.
2. *Multitypes*: The set of multitypes $\mathcal{M}$ consists of all subsets $M \subseteq \mathcal{F}$ of fundamental types, i.e., $\mathcal{M} = \mathcal{P}(\mathcal{F}) = \mathcal{P}(\mathcal{P}(\mathcal{A}) - \emptyset)$.

The names of the different sets of fundamental types stems from the terms *Base Preference* and *Complex Preference* in Preference SQL [KEW11]. According to our calculus [MRE12] the type of an element representing a base (or complex) preference is also a base (or complex) type.

In the relational case the type assertions for a multitype $M \in \mathcal{M}$ evolve to

$$R :: M \quad \Leftrightarrow_{df} \quad R \subseteq \bigcup_{T \subseteq M} D_T$$
$$R :: M^2 \quad \Leftrightarrow_{df} \quad R \subseteq \bigcup_{T \subseteq M} (D_T \times D_T)$$

A natural order on multitypes is given by the inclusion order $\subseteq$. The maximal type is $\mathcal{F}$ and this gives us the typing of the 1 and $\top$ elements:

$$1 :: \mathcal{F}, \quad \top :: \mathcal{F}^2$$

In the algebraic setting the type assertions for general multityped elements can be expressed as

$$a :: M^2 \quad \Leftrightarrow_{df} \quad a \leq \sum_{T \in M} 1_T \cdot a \cdot 1_T \qquad \text{for} \quad M \in \mathcal{M} \; .$$

Note that these assertions allow only relations between tuples of the same fundamental type as the following example shows.

Example 3.2. Assume attributes A, B with type domains $D_A = \{A_1, A_2\}$ and $D_B = \{B_1, B_2\}$. The set $X = \{(A_1, A_2), (B_1, B_2)\}$ fulfils the type assertion

$$X :: \{\{A\}, \{B\}\}^2$$

In contrast the set $Y = \{(A_1, B_1)\}$ does not fulfil the assertion $Y :: \{\{A\}, \{B\}\}^2$ as $Y \notin (D_A \times D_A) \cup (D_B \times D_B)$. This is intended, as inhomogeneous preference relations are not allowed. But note that $Y :: \{A\} \bowtie \{B\}$ is true, because (A_1, B_1) is not interpreted as a preference relation, but as a tuple of a dataset with the attributes A, B, i.e., as contained in a database table with columns A, B.

The set of multitypes $\mathcal{M}$ is the power set of $\mathcal{F}$, hence multitypes are closed under union and intersection. The fundamental types are not contained in $\mathcal{M}$, but any fundamental type $T \in \mathcal{F}$ has a corresponding multitype $\{T\}$. Hence we can generalize the union of arbitrary types, which we will need later on for specifying the type of an addition:

$$T_1 \cup_{\mathrm{m}} T_2 \;=_{df}\; T_1' \cup T_2' \;\text{ where }\; T' := \begin{cases} \{T\} & \text{for } T \in \mathcal{F} \\ T & \text{for } T \in \mathcal{M} \end{cases}$$

Completely analogous to this we define the multitype intersection $\cap_{\mathrm{m}}$.

3.2 Properties of multityped elements

Now we want to study some consequences of the multitype setting and give some useful properties.

Assume an element $a :: T$. By definition of the type assertions, this fulfils also the type assertions $a :: M$ for any element $M \subseteq \mathcal{A}$ with $T \in M$. Hence an element has its "real type" and simultaneously the type of all "supertypes". Something similar happens in object oriented programming where an object is simultaneously the instance of its actual class and all its super classes in the class hierarchy.

The empty set $\emptyset$ is also a multitype, and the only elements fulfilling $x :: \emptyset$ is the smallest element of our algebra, i.e., 0. But still $0 :: T$ holds for all $T \in \mathcal{F}$, additionally $0 :: M$ for all $M \in \mathcal{M}$. To get rid of this ambiguity, we define the minimal type as the smallest multitype which fulfils the type assertion, formally stated in the following definition.

Definition 3.3 (Minimal type). The minimal type for a general element x is defined as follows:

$$x \overset{\min}{::} M^2 \;\;\Leftrightarrow_{df}\;\; M = \bigcap \{N \in \mathcal{M} \mid x :: N^2\}$$

where "min" is w.r.t. to the inclusion order $\subseteq$ on multitypes.

As a consequence of the multitype setting we can determine the resulting type of an addition of arbitrary typed elements.

Corollary 3.4 (Type of an addition). *Let $a :: M_a^2, b :: M_b^2$. Then we have*

$$a + b :: (M_a \cup_{\mathrm{m}} M_b)^2$$

Proof. Assume M_x to be multitypes for $x \in \{a, b\}$. From the type assertions of a and b we have $a \le \sum_{T \in M_a} 1_T \cdot a \cdot 1_T$ and $b \le \sum_{T \in M_b} 1_T \cdot b \cdot 1_T$. By adding both inequations, and using $1_T \cdot a \cdot 1_T = 0$ for $T \notin M_a$ and analogously $1_T \cdot b \cdot 1_T = 0$ for $T \notin M_b$ we conclude:

$$\begin{aligned} a + b &\le \sum_{T \in M_a} 1_T \cdot a \cdot 1_T + \sum_{T \in M_b} 1_T \cdot b \cdot 1_T \\ &\le \sum_{T \in M_a} 1_T \cdot (a+b) \cdot 1_T + \sum_{T \in M_b} 1_T \cdot (a+b) \cdot 1_T = \sum_{T \in M_a \cup_{\mathrm{m}} M_b} 1_T \cdot (a+b) \cdot 1_T \end{aligned}$$

This is the type assertion equivalent to the claim. If M_x is no multitype, consider its corresponding multitype $\{M_x\}$ and the same argument holds. $\square$

In the relational setting this corollary holds also when "$::$" is replaced by "$\overset{\min}{::}$".

Corollary 3.5 (Type of a composition). *Let $a :: M_a^2, b :: M_b^2$. Then we have*

$$a \cdot b :: (M_a \cap_{\mathrm{m}} M_b)^2$$

Proof. Analogously to the proof of Corollary 3.4 assume M_a, M_b to be multitypes. Using $1_T \cdot 1_U = 0$ for $T \neq U$ we conclude

$$\begin{aligned} a \cdot b &\leq \sum_{T \in M_a} 1_T \cdot a \cdot 1_T \cdot \sum_{T \in M_b} 1_T \cdot b \cdot 1_T \leq \sum_{T \in M_a \cap_{\mathrm{m}} M_b} 1_T \cdot a \cdot 1_T \cdot b \cdot 1_T \\ &\leq \sum_{T \in M_a \cap_{\mathrm{m}} M_b} 1_T \cdot a \cdot b \cdot 1_T \end{aligned}$$

which is the claimed type assertion. $\square$

Note that this corollary does not hold if "$::$" is replaced by "$\overset{\min}{::}$" as the composition may be of smaller type. For example the composition of disjoint tests of the same type equals 0, and we have $0 \overset{\min}{::} \emptyset$.

4 Conclusion and further work

We have sketched an extended typing mechanism for our algebra where arbitrary inhomogeneous unions of general elements are covered. This allows also a consistent typing of 1 and $\top$ in the algebra. This is important for the uniformity of the typing mechanism although the inhomogeneous unions have currently no interpretation in our field of application, the database preferences.

But they could be useful for algebraic models of databases, e.g., a set of views of database schemas can be typed by a multitype. Algebraically it is an interesting question how this multitype setting could be extended to more complex algebras, e.g., inhomogeneous relation algebras or elements causing side-effects.

References

[KEW11] W. Kießling, M. Endres, F. Wenzel: The Preference SQL System – An Overview. In IEEE Data Eng. Bull., Vol. 34 (2), 11–18, 2011.

[MR12] B. Möller, P. Roocks: An Algebra of Layered Complex Preferences. To appear in Relational and Algebraic Methods in Computer Science (RAMiCS '12).

[MRE12] B. Möller, P. Roocks, M. Endres: An Algebraic Calculus of Database Preferences. In Mathematics of Program Construction (MPC '12), 241–262, 2012.

[Tha93] B. Thalheim: Foundations of entity–relationship modeling. In Annals of Mathematics and Artificial Intelligence, 197–256, 1993.