Tagging as an alternative to object creation*

Marc Gyssens
University of Limburg
Department WNI
B-3590 Diepenbeek, Belgium
e-mail: gyssens@ccu.uia.ac.be

Lawrence V. Saxton
University of Regina
Department of Computer Science
Regina, Saskatchewan S4S 0A2, Canada
e-mail: saxton@mercury.cs.uregina.ca

Dirk Van Gucht
Computer Science Department
Indiana University
Bloomington, Indiana 47405-4101, USA
e-mail: vgucht@iuvax.cs.indiana.edu

Abstract

Based on the observation that graphs play an important role in the representation of databases, an algebra is presented for the manipulation of binary relations, i.e., of directed unlabeled graphs. This so-called *Tarski algebra* is based on early work by Tarski. The key notion that has been added to it here is *tagging*, which is needed for providing both enough modeling power and enough querying power. Moreover, tagging can also be seen as a value-based counterpart to object creation in object-oriented data models. We present tagging in a general formal framework that incorporates several specific tagging strategies as a special case. We show that each of these strategies allows for the simulation in the Tarski model of various other database models, in particular of the relational model. Finally, we discuss the genericity of tagging and show that the Tarski algebra augmented with multiple assignments and a while-construct is a computationally complete database language.

*Presented at the Dagstuhl Seminar on Query Processing in Object-Oriented, Complex Object, and Nested Relation Databases

1 Introduction

The current trend in database research is towards systems which can support database applications that involve data objects with a complex external and internal organization. This trend is reflected by the emergence of systems which support extendible databases [6, 27, 31] and object-oriented databases [4, 5, 20, 31]. The database theory community has followed this trend and has devised mathematical database models to both represent and manipulate complex-object databases. These database models can be categorized into two general classes: value-based models (the so-called *complex-object database models*) and object-identity-based models (the so-called *object-oriented database models*).

Parent	Child
p_1	p_3
p_1	p_4
p_2	p_3
p_2	p_4
p_4	p_6
p_5	p_6

Person	Hobby
p_1	Soccer
p_1	Music
p_2	Dance
p_3	Soccer
p_4	Reading
p_4	Painting
p_5	Basketball
p_6	Music
p_6	Games

Person	Name	Age
p_1	Frank	50
p_2	Ellen	52
p_3	Eric	30
p_4	Lisa	28
p_5	Brad	31
p_6	Eric	6

Figure 1: A persons relational database

To illustrate the difference between these two classes, consider the persons relational database shown in Figure 1. This database could alternatively be represented as a complex-object database as shown in Figure 2. Although transforming a flat relational database into a complex-object database of course requires the creation of complex objects (i.e., the name-age pairs, the children sets, and the hobbies sets in our example), it is important to notice that *no* new atomic values are introduced.

In contrast, an object-oriented representation requires the introduction of *new* atomic values in the form of object-identifiers of objects. For example, one possible way of representing the persons database in an object-oriented context is shown in Figure 3. (For simplicity, we used an untyped representation rather than a more traditional, i.e., class-oriented approach.) Each object carries its own value and identity, which in turn is allowed to occur in the value components of other objects. Consider for example the object with object-identifier s_1. This object-identifier uniquely represents a children set. Its value is the set of children of the parent objects identified by o_1 and o_2. Notice how s_1 is *shared* in the value components of the respective parents.

It would appear from this example that complex-object databases and object-oriented databases are fundamentally different, each offering its own advantages. The main advantage of the complex-object approach is that no new atomic values are introduced. This provides a clean framework in which to extend the theory of relational databases to that of databases with complex objects. On the other hand, the main advantage of the object-oriented approach is its economy of representation. For example, whereas in the object-oriented version

Person	Personal-Data	Children	Hobbies
p_1	[name: Frank, age: 50]	Child: p_3, p_4	Hobby: Soccer, Music
p_2	[name: Ellen, age: 52]	Child: p_3, p_4	Hobby: Dance
p_3	[name: Eric, age: 30]	Child: p_6	Hobby: Soccer
p_4	[name: Lisa, age: 28]	Child: p_6	Hobby: Reading, Painting
p_5	[name: Brad, age: 31]	Child	Hobby: Basketball
p_6	[name: Eric, age: 6]	Child	Hobby: Music, Games

Figure 2: The person database represented as a complex-object database

Object Identifier	Object Value
o_1	[id: p_1, info: i_1, children: s_1,] hobbies: {Soccer, Music}]
o_2	[id: p_2, info: i_2, children: s_1, hobbies: {Dance}]
o_3	[id: p_3, info: i_3, children: s_2, hobbies: {Soccer}]
o_4	[id: p_4, info: i_4, children: s_3, hobbies: {Reading, Painting }]
o_5	[id: p_5, info: i_5, children: s_3, hobbies: {Basketball}]
o_6	[id: p_6, info: i_6, children: s_2, hobbies: {Music, Games}]
i_1	[name: Frank, age: 50]
i_2	[name: Ellen, age: 52]
i_3	[name: Eric, age: 30]
i_4	[name: Lisa, age: 28]
i_5	[name: Brad, age: 31]
i_6	[name: Eric, age: 6]
s_1	$\{o_3, o_4\}$
s_2	$\emptyset$
s_3	$\{o_6\}$

Figure 3: The person database represented as a object-oriented database

of the person database, there is a unique representation for the children set of o_1 and o_2, this children set needs to be doubly represented in the complex-object persons database. This economy of representation, however, comes at the expense of being able to extend cleanly the theory of relational databases to that of object-oriented databases.

Parents	Children
τ_3	τ_1
τ_4	τ_2

Tag	Value
τ_1	$\{p_3, p_4\}$
τ_2	$\{p_6\}$
τ_3	$\{p_1, p_2\}$
τ_4	$\{p_4, p_5\}$

Figure 4: The parent-child relation using tags

To bridge the gap between complex-object databases and object-oriented databases, researchers have proposed database models and database languages in which new values can be introduced through object-creation. For example, returning to the persons database, assume that we have derived at some stage in a computation that the sets $\{p_3, p_4\}$ and $\{p_6\}$ are the children-sets of the (unordered) parent pairs $\{p_1, p_2\}$ and $\{p_4, p_5\}$, respectively. Instead of carrying the full representation of this derived information throughout the rest of the computation, we could decide to *create* four *new* values, say τ_1, τ_2, τ_3, and τ_4. These values uniquely represent the respective children sets and parent pairs. Using these values, the derived parent-child information can be represented as shown in Figure 4.

In this paper, we will call these new values *tags*. It must be emphasized that tags are *different* from object-identifiers. Tags can be thought of as succinct representations of complex objects, whereas object-identifiers are uniquely associated with objects whose value can be a complex object.

Indeed, turning back to our example, suppose that p_1 and p_2 have a new child, say p_7. In the complex-object database, this update would be reflected through the addition of p_7 to the children sets of the respective parents p_1 and p_2. In the object-oriented approach, however, p_7 would simply be added as a new object (say with object identifier o_7), and the value of the object s_1 would be updated to include o_7. In particular, s_1 does not have to be replaced throughout the database. In the tag-based approach in contrast, the only reasonable strategy is to replace the tag τ_1 of the persons set $\{p_3, p_4\}$ by a new tag, say τ_5, representing the set $\{p_3, p_4, p_7\}$. In other words, after this update, τ_1 is *still* the representation of the "old" set $\{p_3, p_4\}$.

For this reason, the tagging technique is essentially value-based. However, at the same time, economy of representation and expression is obtained.

There exist essentially two classes of tag-based database models. The first class, which we call the class of *non-deterministic* tag-based database models, consists of database models which introduce tags non-deterministically. The non-determinism stems from the idea that it does not matter which new values are actually introduced to represent complex objects. Of course, the consequence of the non-deterministic approach is that there may be several tags, i.e., *copies*, representing the same complex object. The classic example of a non-deterministic tag-based database model is the the IQL model of Abiteboul and Kanellakis [1]. In the same class are also transactions-based database models of Abiteboul and Vianu [2] and the GOOD model of Gyssens et al. [14, 15]. The second class, which we call the class of *deterministic* tag-based database models, consists of database models which introduce new tags deterministically through function evaluations of complex objects (this process is closely related to the notion of skolemization in logic). In this class, one finds LDM [23], C-logic [8] and F-logic [19]. In the ILOG model of Hull and Yoshikawa [18], both the non-deterministic and the deterministic approach are combined. In the limit of course, any complex-objects database model can be thought of as deterministic tag-based database model, simply because an object-value can always be thought of as its own tag (such an approach was actually used for theoretical purposes in [13]).

Most tag-based database models have been proposed in a declarative, logic-based context, with the GOOD model and LDM being the exceptions. In contrast with the satisfactory status of declarative formulations of tag-based database models, we feel that there are currently few, if any, satisfactory algebraic formulations. This is unfortunate since algebraic formulations are better platforms from which to build real database systems. In this paper we offer such an algebraic formulation. We will introduce a tag-based database model, the *Tarski database model*. We called our model, based on binary relations, after the distinguished logician-mathematician Tarski who laid the set-theoretic foundations for our proposal.

Our model is an outgrowth of a mechanism to algebratize the GOOD model. To understand this, it is necessary to discuss briefly the GOOD database model and its associated data manipulation language. This will be the subject of Section 2. In Section 3, we then formally introduce the Tarski database model. Our main emphasis will be on the data manipulation language, which we called the *Tarski algebra*. In Section 4, we show how other database models (relational model, nested model, GOOD model) can be simulated using the essentially binary Tarski approach. In particular, we show that the Tarski algebra allows the expression of the relational algebra. In Section 5, we extend the Tarski algebra with a looping construct and show that this yields a computationally complete language. In this section, we

also discuss the notion of genericity in the presence of tags. Finally in Section 6, we mention some ongoing and future research on the Tarski database model, including implementation issues.

2 Towards a tag-based database model

2.1 The Graph-Oriented Object Database model (GOOD)

The Graph-Oriented Object Database model (GOOD) was introduced in [14, 15] as an attempt to construct an object-oriented database model in which both the data representation part and the data manipulation language are graph-based.

Figure 5: The persons database as a GOOD instance

In the GOOD model, a database is viewed as a labeled directed graph. For example, in Figure 5 we show a GOOD database for the persons database shown in Figure 1. For simplicity, we omitted the information about age and hobbies. Circle nodes represent printable information, while square nodes can easily be interpreted as tags for complex objects. Ordinary arrows represent functional properties whereas double arrows indicate multivalued properties.

Data manipulation operations in GOOD are specified in term of graph-transformations such as node additions, edge additions, node deletions and edge deletions. For example, in

Figure 6: Some GOOD operations on the parents database

Figure 7: The effect of the operations in Figure 6 on the persons database

Figure 6 *(left)* we show a node addition operation the effect of which is to add new parent-pair nodes, i.e., one new such node for each pair of person nodes which have a common child. (In the context of our discussion of tags, think of these new nodes as tags for the respective parent-pairs.) In Figure 6 *(middle)* we show an edge addition operation. Its effect is to associate the children nodes directly with their corresponding parent-pair nodes. Finally, in Figure 6 *(right)* we show an edge deletion that removes the original links between parents and children. The effect of the consecutive application of these three operations is displayed in Figure 7.

2.2 GOOD as motivation for a binary tag-based database model

A natural way to represent a GOOD database algebraically is as a collection of mathematical (binary) relations [24]. In one particular such representation there are two relations to represent the nodes in the GOOD database, i.e, one associating node labels to nodes and one associating print labels to printable nodes. Furthermore, there is a separate relation for each edge-label occurring in the GOOD database, i.e, if e is such an edge label, then e can be represented by a relation and a pair (n_1, n_2) in this relation denotes that there is an edge in the GOOD database with label e from node n_1 to node n_2. Figure 8 shows this representation for the portion of the persons database represented in GOOD in Figure 5.

Of course, one can easily think of other reasonable ways to represent GOOD databases in terms of mathematical relations.

Once a particular representation has been achieved, one can simulate GOOD graph transformations by set-theoretic operations on mathematical relations.

As it turns out, Tarski [28] had already specified an algebra to manipulate just such binary relations. In fact, Tarski called this algebra a *relation algebra* in later work. However, he did not use it to manipulate databases, but rather to study set-theoretical concepts. Tarski's algebra has (only) four fundamental and well-known operators: union, composition, complementation, and inversion. Furthermore, Tarski assumes that there is a special relation

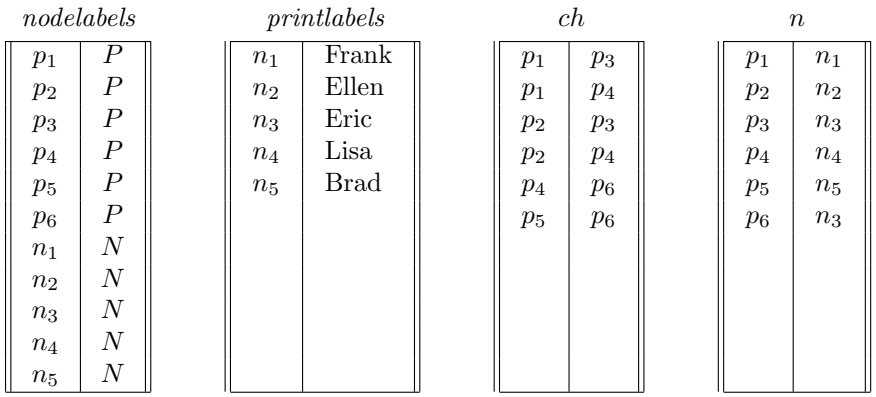


Figure 8: Using binary mathematical relations for representing GOOD instances

fixing the equality relation on the universe of discourse. In a very real sense, mathematicians and logicians were already doing relational algebra well before Codd introduced his relational algebra into the context of databases. Of course mathematicians almost invariantly dealt with infinite relations, while database theoreticians, and certainly database practitioners, worked with finite relations.

In this paper we approach database theory following Tarski's rather than Codd's formalism.[1] We model a database by a set of mathematical relations and manipulate a database by performing Tarski algebraic operations on these relations. In order to study the notion of tagging, we add two new operators to Tarski's algebra. These so-called *tagging operators* allow the introduction of (new) tags. Each new tag represents a specific ordered pair in the database and is derived through a so-called *tagging function* evaluation. The power of the tagging technique stems from the fact that these new tag values can be used as values in other pairs, thus facilitating the succinct representation of complex objects. In combination with the other operators of the Tarski algebra, the tagging operators allow for complex database manipulations.

We will furthermore show that tagging can be achieved via tagging functions with special properties. These properties, we believe, are in fact effectively used in real-world database systems. So our contribution is two-fold: one, we give a simple algebraic formulation of tag-based database models, and two, our proposed tagging techniques correspond naturally to techniques already in use in database systems.

[1]We are not the first to do this. Kraegeloh and Lockemann [22] already did so in 1975. The main difference between their proposal and ours is that they do not consider the additions of new values to their database. As was demonstrated by Tarski and Givant [29], the expressive power of the data manipulation language then is very limited.

3 The Tarski algebra

3.1 Codd relations and mathematical relations

Throughout this paper, we assume infinitely enumerable sets U and V of *attributes* and *values*, respectively.

Using the formalism of [13], *Codd scheme* is a finite subset of U. Let Ω be a Codd scheme. A tuple over Ω is a mapping of Ω into V; a *Codd instance* over Ω is a set of tuples over Ω. All Codd instances are considered to be finite unless explicitly stated otherwise. Finally, a *Codd relation* is a pair (Ω, ω) with Ω a Codd scheme and ω a Codd instance over Ω.

In contrast to Codd relations, the building blocks of the relational database model [9, 10, 11], mathematical relations are sets of ordered pairs of values. For our purposes, a mathematical relation (or *relation*, for short) is a subset of $V \times V$. Of course, there is a close connection between a mathematical relation and a Codd relation with two attributes but, unlike the latter, the former does not have a scheme.

In what follows, we will develop tools to manipulate relations in the mathematical sense and show how these techniques apply to various database models. All relations are considered to be finite unless explicitly stated otherwise.

It is often important to know the values actually involved in a given relation r. We denote by $r{\downarrow}$ the smallest subset of V such that $r \subset r{\downarrow} \times r{\downarrow}$, i.e., $r{\downarrow} = \{v \in V \mid \exists w \in V : (v, w) \in r \vee (w, v) \in r\}$. The set $r{\downarrow}$ will be called the *active domain* of r. Similarly, if $r_1, \ldots, r_m$ are relations, then $(r_1, \ldots, r_m){\downarrow} = r_1{\downarrow} \cup \ldots \cup r_m{\downarrow}$ is the active domain of the database consisting of the mathematical relations $r_1, \ldots, r_m$.

3.2 The basic algebra on relations

Mathematical relations, because of their natural affinity with graphs, are very important and fundamental data structures in several areas of computer science and, therefore, it is important to have appropriate tools to manipulate them. Because of the close connection between mathematical relations and Codd relations with two attributes, it seems natural to borrow these tools from relational algebra. However, long before Codd devised his relational algebra in 1970 [9], Tarski already had defined an algebra on mathematical relations [28]. The operators defined below are inspired by Tarski's algebra.

Definition 3.1 *Let r and s be relations. We define the following operations:*

- *The* inverse *of r, denoted r^{-1}, is the relation $\{(v, w) \mid (w, v) \in r\}$.*
- *The* complement *of r, denoted $\overline{r}$, is the relation $\{(v, w) \mid (v, w) \in r{\downarrow} \times r{\downarrow} \wedge (v, w) \notin r\}$.*
- *The* union *of r and s, denoted $r \cup s$, is the relation $\{(v, w) \mid (v, w) \in r \vee (v, w) \in s\}$.*
- *The* composition *of r and s, denoted $r \cdot s$, is the relation $\{(u, w) \mid \exists v \in V : (u, v) \in r \wedge (v, w) \in s\}$.*

A major difference between the operators defined above and their counterparts in Tarski's algebra is that we define complementation relative to the active domain rather than to the

set V of all values, as Tarski does. Our approach of course aims at avoiding generating infinite relations.

In order to introduce a *basic algebra expression (bae)* we assume an infinitely enumerable set of (relation) variables $x_1, x_2, x_3, \ldots$ Bae's are then recursively defined as follows:

1. $\emptyset$ is a bae;
2. each variable x_i is a bae; and
3. an expression obtained by applying the operators of Definition 3.1 to bae's is again a bae.

In writing down expressions, we assume that unary operators take precedence over binary operators; also, composition takes precedence over set-like operators. If $E(x_1, \ldots, x_m)$ is a bae and $r_1, \ldots, r_m$ are relations, then the interpretation of $E(r_1, \ldots, r_m)$ is defined in the customary way. The set of all bae's will be called the *basic algebra*.

3.3 Tagging

We will now formally introduce the concept of *tagging*. Intuitively, we view tagging as attaching new values (*tags*) to objects of the database with the purpose of identifying them. Tagging is a technique widely used in the field of databases, e.g., in order to manipulate nested relations or to introduce object identifiers in object-oriented databases. In this subsection, we will study the notion of tagging in the context of mathematical relations. Later on, we will discuss tagging in the context of relations, nested relations and object-oriented databases.

In order to introduce tagging, we have to assume that the set of values V consists of an infinite set D of *data values* and an infinite set T of *tag values*, i.e., $V = D \cup T$ with $D \cap T = \emptyset$. The set D consists of these values that actually represent "data" of our database, while T is a set of "auxiliary" values, needed to introduce tags.

In the context of mathematical relations, the "objects" we want to tag are, quite straightforwardly, the ordered pairs. Before we can proceed to a formal definition however, we first have to realize that several variations on the idea of tagging are conceivable, most of which are actually around in database practice. To illustrate our point, consider the relations shown in Figure 9.

r

a	b
a	c
b	c
c	d
c	c

s

c	a
b	a
a	b

Figure 9: Two mathematical relations

A first tagging strategy we wish to discuss will be called *tuple tagging*. In this strategy, a "generic" tag is associated to each ordered pair, e.g., by writing its components between square brackets, as shown in Figure 10. A non-operational variation on tuple tagging has already been considered in [29].

r

$[a, b]$	a	b
$[a, c]$	a	c
$[b, c]$	b	c
$[c, d]$	c	d
$[c, c]$	c	c

s

$[c, a]$	c	a
$[b, a]$	b	a
$[a, b]$	a	b

Figure 10: Tuple tagging

A consequence of this approach is that a same pair always gets the same tag, irrespective of the relation in which it occurs. In Figure 10 for instance, the ordered pair (a, b) obtains the same tag in both r and s. One could easily conceive situations in which the tag should not only depend on the ordered pair but also on the relation in which the ordered pair occurs. This latter tagging strategy will be called *relation tagging*. The assignment of tags in Figure 11 is consistent with a relation tagging strategy.

r

1	a	b
2	a	c
3	b	c
4	c	d
5	c	c

s

6	c	a
7	b	a
8	a	b

Figure 11: Relation tagging

In the relation tagging strategy it is still possible that a same value is introduced multiple times, e.g., when a same relation needs to be tagged twice in the course of solving a query. The tagging strategy in which no value is introduced more than once as a tag will be called *universal tagging*.

Rather than making a choice between one of these various alternatives, we preferred to introduce a more general notion of tagging which encompasses all three strategies outlined above as special cases.

To define this generalized tagging, let $\mathcal{V}$ be the set of all pairs (s, t) with s a relation (i.e., a finite subset of $V \times V$) and $t = (v, w)$ an ordered pair over V. (s will be called a *context*). A *tagging function* τ is a bijection:

$$\tau : \mathcal{V} \longrightarrow T$$

(T being the set of tag values) such that $\tau(s, t)$ does *not* occur in $s \cup \{v, w\}$. Intuitively, τ associates a tag value to the ordered pair t that depends on the context s. Moreover, this

tag must be "new", i.e., it can not be a component of the pair t and it must not already occur in the context s.

The various tagging strategies discussed above can now easily be obtained by making a particular choice for the context.

If we systematically choose the empty relation $\emptyset$ as the tagging context, we achieve tuple tagging. If, alternatively, we take for the context the relation in which the pair under consideration occurs, we get relation tagging. Finally, if the context is a representation of the set of all values that up to then did occur at some point in the database under consideration, then we achieve universal tagging. Most of the results we prove in this paper hold independent of the particular tagging strategy chosen. This will become important in Section 5.3, where we discuss the expressive power of the language we are presently defining.

Given a tagging function, we now introduce two tagging operators:

Definition 3.2 *Let r and s be relations.*

- *The* left-tagging *of r in the context of s, denoted $r_s^{\triangleleft}$, is the relation*
$$\big\{\big(\tau(s,(v,w)),v\big)\big|(v,w)\in r\big\}$$
.
- *The* right-tagging *of r in the context s, denoted $r_s^{\triangleright}$, is the relation*
$$\big\{\big(\tau(s,(v,w)),w\big)\big|(v,w)\in r\big\}$$
.

Observe that the relation r can always be reconstructed from its left- and right-tagging whatever context was used, provided it was the same for both operation. Indeed, left- and right-tagging always satisfy the property:

$$r = (r_s^{\triangleleft})^{-1}\cdot r_s^{\triangleright}$$

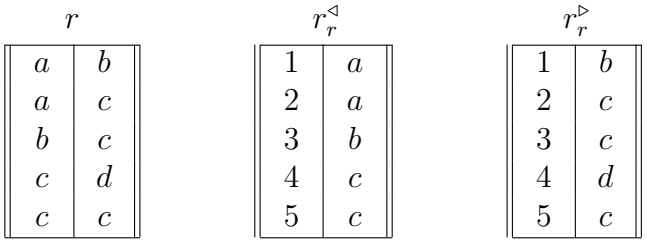
As data values and tag values are disjoint ($D\cap T=\emptyset$), the actual values of the tags are immaterial; only their equality or difference is of relevance. Therefore, in concrete examples, rather than specifying a tag function which would be quite cumbersome a job, we shall use (consecutive) integers to represent tag values; data values will be represented by lower case letters.

Example 3.1 In Figure 12, we show the result of left and right tagging a relation in its own context.

The algebra obtained by augmenting the basic algebra with the left- and right-tagging operators will be called the *Tarski algebra*. An expression of the Tarski algebra will be called a *Tarski algebra expression (tae)* and defined in analogy to a bae.

The Tarski algebra is considerably more powerful than the basic algebra. To illustrate this, we show that:

Proposition 3.1 *Let r be a relation over V. The* identity operator
$$r^{\mathrm{id}} = \{(v,w)\in r{\downarrow}\times r{\downarrow}\mid v=w\}$$
can be expressed in the Tarski algebra.



r	
a	b
a	c
b	c
c	d
c	c

$r_r^{\triangleleft}$	
1	a
2	a
3	b
4	c
5	c

$r_r^{\triangleright}$	
1	b
2	c
3	c
4	d
5	c

Figure 12: Example of tagging

Proof: Let s be an arbitrary relation. The proposition follows from the equality:

$$r^{\mathrm{id}} = (r_s^{\triangleleft})^{-1} \cdot r_s^{\triangleleft} \cup (r_s^{\triangleright})^{-1} \cdot r_s^{\triangleright}$$

□

Clearly, the identity operator is not expressible in the basic algebra. Several auxiliary operators can be expressed using the identity operator:

Proposition 3.2 *Let r and s be relations. The following operations can be expressed in the basic algebra plus the identity operator:*

- *the* diversity operator $r^{\mathrm{di}} = \{(v, w) \in r\downarrow \times r\downarrow \mid v \neq w\}$*;*
- *the* relative universe $r^{\mathrm{un}} = r\downarrow \times r\downarrow$*;*

Proof: The proposition follows from the equalities:

$$r^{\mathrm{di}} = \overline{r^{\mathrm{id}}}$$
$$r^{\mathrm{un}} = r^{\mathrm{id}} \cup r^{\mathrm{di}}$$

□

Another useful operator is the intersection operator. We show that intersection can also be expressed in the Tarski algebra. While the proof is based on the well-known set-theoretic property relating intersection, union, and complementation, some care is needed since, unlike in set theory, our notion of complementation is relative to the set of values occurring in the relation under consideration.

Proposition 3.3 *Let r and s be relations over V. The* intersection operator $r \cap s = \{(v, w) \mid (v, w) \in r \wedge (v, w) \in s\}$ *can be expressed in the Tarski algebra.*

Proof: Let $(r \cap s)^{=}$ and $(r \cap s)^{\neq}$ be the identical respectively the non-identical tuples of $r \cap s$. The reader is invited to verify that

$$(r \cap s)^{\neq} = \overline{\overline{r \cup (r \cup s)^{\mathrm{id}}} \cup \overline{s \cup (r \cup s)^{\mathrm{id}}} \cup (r \cup s)^{\mathrm{id}}} \tag{1}$$

Now let q be an arbitrary relation and let $p = (r \cup s)^{\triangleright}_q \cdot \left((r \cup s)^{\triangleright}_q\right)^{-1}$. Then

$$(r \cap s)^{=} = \overline{\overline{r \cup (r \cup s \cup p)^{\mathrm{di}}} \cup \overline{s \cup (r \cup s \cup p)^{\mathrm{di}}} \cup (r \cup s \cup p)^{\mathrm{di}}} \tag{2}$$

An expression for $r \cap s$ is now obtained by taking the union of the expressions displayed in (1) and (2). □

In order to understand better the previous proof it should be noted that whenever $(r \cup s)\downarrow$ is a singleton, $(r \cup s)^{\mathrm{di}} = \emptyset$ due to the relative complementation. Hence $(r \cup s)^{\mathrm{di}}$, contrary to $(r \cup s)^{\mathrm{id}}$, does not always contain all values present in r or s, and for that reason the equality

$$(r \cap s)^{=} = \overline{\overline{r \cup (r \cup s)^{\mathrm{di}}} \cup \overline{s \cup (r \cup s)^{\mathrm{di}}} \cup (r \cup s)^{\mathrm{di}}}$$

does *not* always hold. If r and s are both the same singleton relations consisting of an identical pair, then the above expression yields the empty relation rather than the singleton relation. Since a tag to a pair by definition cannot be a value of that pair, $(r \cup s \cup p\downarrow)$ contains *two* values whenever $(r \cup s)\downarrow$ is a singleton. Hence $(r \cup s \cup p)^{\mathrm{di}}$ *always* contains all values present in r, s or p, whence equality (2) in the proof holds.

An alternative and much shorter proof follows from the equality

$$r \cap s = (r^{\triangleleft}_q)^{-1} \cdot s^{\triangleright}_q \tag{3}$$

with q an arbitrary relation. The simplicity of the latter equality is due to the fact that associating a tag to a *same* tuple in a *same* context, yields the *same* tag value. We preferred however to exhibit an expression which is not "context-sensitive", proving that all tagging strategies presented thus far allow the expression of the intersection operator. Notice that although equality (3) can be used to show that intersection is expressible using the tuple tagging strategy (put $q = \emptyset$), it does not show that intersection is also expressible using the relation tagging strategy.

As a corollary to Proposition 3.3, we have:

Proposition 3.4 *Let r and s be relations. The following operations can be expressed in the Tarski algebra:*

- *the* identity selection $r^{=} = \{(v, w) \in r \mid v = w\}$*;*
- *the* diversity selection $r^{\neq} = \{(v, w) \in r \mid v \neq w\}$*; and*
- *the* difference $r - s = \{(v, w) \mid (v, w) \in r \wedge (v, w) \notin s\}$.

Proof: The proposition follows from the equalities:

$$\begin{aligned} r^{=} &= r^{\mathrm{id}} \cap r \\ r^{\neq} &= r^{\mathrm{di}} \cap r \\ r - s &= r \cap (\overline{s \cup r^{\mathrm{id}}} \cup \overline{s \cup r^{\mathrm{di}}}) \end{aligned}$$

□

As a final example illustrating the power of the Tarski algebra, we formally introduce conditional queries:

Definition 3.3 *A* conditional query *is an expression of the form*

$$\begin{array}{l}\textbf{if } E_1(\vec{x}_1) = \emptyset \\ \quad \textbf{then} \\ \quad\quad E_2(\vec{x}_2) \\ \quad \textbf{else} \\ \quad\quad E_3(\vec{x}_3)\end{array}$$

with $E_1(\vec{x}_1)$, $E_2(\vec{x}_2)$ and $E_3(\vec{x}_3)$ arbitrary Tarski algebra expressions. ($\vec{x}_1$, $\vec{x}_2$ and $\vec{x}_3$ denote finite sequences of (mathematical) relation variables.)

We can show:

Proposition 3.5 *The conditional expression Q can be expressed in the Tarski algebra.*

Proof: Consider the following expression:

$$\begin{aligned} E_4(\vec{x}_1, \vec{x}_2, \vec{x}_3) \quad = \quad & E_3(\vec{x}_3) \cdot \big(E_1(\vec{x}_1) \cdot (E_1(\vec{x}_1) \cup E_3(\vec{x}_3))^{\mathrm{un}}\big)^{\mathrm{id}} \\ & \cup E_2(\vec{x}_2) - E_2(\vec{x}_2) \cdot \big(E_1(\vec{x}_1) \cdot (E_1(\vec{x}_1) \cup E_2(\vec{x}_2))^{\mathrm{un}}\big)^{\mathrm{id}} \end{aligned}$$

Clearly, for any three relation r_1, r_2, r_3, $Q(r_1, r_2, r_3) = E_4(r_1, r_2, r_3)$. □

4 Simulating other database models

In this section, we intend to show that binary relations equipped with the Tarski algebra suffice to simulate many other languages. In particular, we shall show that the relational model, the nested model and the GOOD model, together with their respective basic languages, can be expressed in the Tarski model.

The Tarski algebra turns out to be too weak however to express more complete extensions of these languages. Therefore a possible extension of the Tarski algebra dealing with this problem will be discussed in Section 5.

4.1 Relational model

We first turn to the relational model [9, 10, 11], in which a database consists of a set of *Codd relations* (see Subsection 3.1). A Codd relation can be represented by a set of mathematical binary relations as follows:

Definition 4.1 *Let (Ω, ω) be a Codd relation over V with $\Omega = \{A_1, \ldots, A_n\}$. A set of mathematical relations $\{r_{A_1}, \ldots, r_{A_n}\}$ is called a* representation *of (Ω, ω) if there exists a one to one mapping ρ from ω into T, called a* representation function*, such that for all $i = 1, \ldots, n$, $r_{A_i} = \{(\rho(t), t(A_i)) \mid t \in \omega\}$.*

Using this encoding of Codd relations into mathematical relations, we can now give a natural meaning to simulating a relational query in the Tarski algebra:

Definition 4.2 *Let $E(x_1, \ldots, x_m)$ be an expression in the relational algebra in which x_i, $1 \leq i \leq m$, is a variable representing a Codd-relation of arity n_i. We say that $E(x_1, \ldots, x_m)$ can be* simulated *in the Tarski algebra if there exists a tae $E'(y_{11}, \ldots, y_{1n_1}, \ldots, y_{m1}, \ldots, y_{mn_m})$ such that for all Codd relations $(\Omega_1, \omega_1), \ldots, (\Omega_m, \omega_m)$ for which $E((\Omega_1, \omega_1), \ldots, (\Omega_m, \omega_m))$ exists, and for all mathematical relations $r_{11}, \ldots, r_{1n_1}, \ldots, r_{m1}, \ldots, r_{mn_m}$ for which the set $\{r_{i1}, \ldots, r_{in_i}\}$ is a representation of (Ω_i, ω_i) for all i, $1 \leq i \leq m$, $E'(r_{11}, \ldots, r_{1n_1}, \ldots, r_{m1}, \ldots$ $\ldots, r_{mn_i})$ is a representation of $E((\Omega_1, \omega_1), \ldots, (\Omega_n, \omega_m))$.*

We have:

Theorem 4.1 *The relational algebra can be simulated in the Tarski algebra.*

Proof: We show how each of the relational algebra operators can be simulated in the Tarski algebra. We shall develop the proof in such a way that it will become obvious that each of the tagging strategies discussed in Subsection 3.3 suffices to achieve the simulations. To this end, p and q will denote arbitrary mathematical relations throughout this proof.

In order not to obscure the argument with heavy notation, we shall limit the number of attributes in the relations under consideration. The generalization of our argument, however, will be obvious.

Projection

Let (Ω, ω) be a relation with $\Omega = \{A, B, C\}$ and let $\{r_A, r_B, r_C\}$ be a representation of (Ω, ω). Consider the projection $\pi_{\{A,B\}}(\Omega, \omega)$ of $\{\Omega, \omega\}$ onto $\{A, B\}$. Unfortunately $\{r_A, r_B\}$ in general is *not* a representation of $\pi_{\{A,B\}}(\Omega, \omega)$, simply because some tuples of this projection may be multiply represented. In order to overcome this problem, we "retag" r_A and r_B as follows.

First, consider the relation $r'_A = r_A^{-1} \cdot r_A$. This relation consists of all identical pairs of A-values occurring in ω. Now let

$$s_A = (r'_A)_p^{\triangleright} \cdot r_A^{-1}$$

Each tag in s_A corresponds to *precisely one* tuple of $\pi_{\{A\}}(\Omega, \omega)$. Moreover, the relation s_A gives the relationship between the "new" and the "old" tags.

Now, let $r'_B = s_A \cdot r_B$ and let

$$s_B = (r'_B)_q^{\triangleright} \cdot r_B^{-1} \cap (r'_B)_q^{\triangleleft} \cdot s_A$$

Each tag in s_B corresponds to *precisely one* tuple of $\pi_{\{A,B\}}(\Omega, \omega)$. Moreover, the relation s_B gives the relationship between the newly created tags and the original ones. Hence $\{s_B \cdot r_A, s_B \cdot r_B\}$ is a representation of $\pi_{\{A,B\}}(\Omega, \omega)$.

Selection

Let (Ω, ω) be a relation with $\Omega = \{A, B, C\}$ and let $\{r_A, r_B, r_C\}$ be a representation of (Ω, ω). Consider the selection $\sigma_{A=B}(\Omega, \omega)$ of $\{\Omega, \omega\}$.

Let $r = (r_A \cdot r_A^{-1})^{\mathrm{id}}$ be the set of all identical pairs of "tags" used in the representation of (Ω, ω). Let $r_{A,B} = r_A \cdot r_B^{-1} \cap r$. The relation $r_{A,B}$ is the set of all identical pairs of tags corresponding to tuples t of ω with $t(A) = t(B)$. Hence, $\{r_{A,B} \cdot r_A,\ r_{A,B} \cdot r_B,\ r_{A,B} \cdot r_C\}$ is a representation of $\sigma_{A=B}(\Omega, \omega)$.

Union

Let (Ω, ω_1) and (Ω, ω_2) be relations with $\Omega = \{A, B\}$. Let $\{r_{1,A}, r_{1,B}\}$ and $\{r_{2,A}, r_{2,B}\}$ be representations of (Ω, ω_1) and (Ω, ω_2), respectively.

The proof for this case is quite similar to the proof for the case of projection.

First, consider the relation $r'_A = r_{1,A}^{-1} \cdot r_{1,A} \cup r_{2,A}^{-1} \cdot r_{2,A}$ consisting of all identical pairs of A-values occurring in ω_1 or ω_2. Let

$$s_{1,A} = (r'_A)_p^{\triangleright} \cdot r_{1,A}^{-1}$$
$$s_{2,A} = (r'_A)_p^{\triangleright} \cdot r_{2,A}^{-1}$$

Each tag value introduced above corresponds to *precisely one* tuple of $\pi_{\{A\}}(\Omega, \omega_1 \cup \omega_2)$. Moreover, $s_{1,A}$ and $s_{2,A}$ give the relationship between the "new" and the "old" tags.

Now, let $r'_B = s_{1,A} \cdot r_{1,B} \cup s_{2,A} \cdot r_{2,B}$ and let

$$s_{1,B} = (r'_B)_q^{\triangleright} \cdot r_{1,B}^{-1} \cap (r'_B)_q^{\triangleleft} \cdot s_{1,A}$$
$$s_{2,B} = (r'_B)_q^{\triangleright} \cdot r_{2,B}^{-1} \cap (r'_B)_q^{\triangleleft} \cdot s_{2,A}$$

Each tag value introduced above corresponds to *precisely one* tuple of $(\Omega, \omega_1 \cup \omega_2)$. Moreover, $s_{1,A}$ and $s_{2,A}$ give the relationship between the newly created tags and the original ones. Hence $\{s_{1,B} \cdot r_{1,A} \cup s_{2,B} \cdot r_{2,A},\ s_{1,B} \cdot r_{1,B} \cup s_{2,B} \cdot r_{2,B}\}$ is a representation of $(\Omega, \omega_1 \cup \omega_2)$.

Difference

Let (Ω, ω_1) and (Ω, ω_2) be relations with $\Omega = \{A, B, C\}$. Let $\{r_{1,A}, r_{1,B}, r_{1,C}\}$ and $\{r_{2,A}, r_{2_B}, r_{2,C}\}$ be representations of (Ω, ω_1) and (Ω, ω_2), respectively. Let

$$r = r_{1,A} \cdot r_{2,A}^{-1} \cap r_{1,B} \cdot r_{2,B}^{-1} \cap r_{1,C} \cdot r_{2,C}^{-1}$$

Let ρ_1 and ρ_2 be the representation function used for representing (Ω, ω_1) respectively (Ω, ω_2). Then $r = \big\{\big(\rho_1(t), \rho_2(t)\big) \big| t \in \omega_1 \cap \omega_2\big\}$. Now let $s = (r_{1,A} \cdot r_{1,A}^{-1})^{\mathrm{id}} - (r \cdot r^{-1})^{\mathrm{id}}$. Then $s = \big\{\big(\rho_1(t), \rho_1(t)\big) \big| t \in \omega_1 - \omega_2\big\}$. Hence $\{s \cdot r_A,\ s \cdot r_B,\ s \cdot r_C\}$ is a representation of $(\Omega_1, \omega_1 - \omega_2)$.

Cartesian product

Finally, let us consider the cartesian product $(\Omega_1 \cup \Omega_2, \omega_1 \times \omega_2)$ of two relations (Ω_1, ω_1) and (Ω_2, ω_2), $\Omega_1 \cap \Omega_2 = \emptyset$, with $\Omega_1 = \{A, B, C\}$ and $\Omega_2 = \{D, E\}$. Let $\{r_A, r_B, r_C\}$ be a representation of (Ω_1, ω_1) and let $\{r_D, r_E\}$ be a representation of (Ω_2, ω_2). Let $r_1 = (r_A \cdot r_A^{-1})^{\mathrm{id}}$, $r_2 = (r_D \cdot r_D^{-1})^{\mathrm{id}}$ and $r = (r_1 \cup r_2)^{\mathrm{un}}$. Let $s = r_1 \cdot r \cdot r_2$. Then $s = \big\{\big(\rho_1(t_1), \rho_2(t_2)\big) \big| t_1 \in \omega_1 \wedge t_2 \in \omega_2\big\}$. Hence $\{s_p^{\triangleleft} \cdot r_A,\ s_p^{\triangleleft} \cdot r_B,\ s_p^{\triangleleft} \cdot r_C,\ s_p^{\triangleright} \cdot r_D,\ s_p^{\triangleright} \cdot r_E\}$ is a representation of $(\Omega_1 \cup \Omega_2, \omega_1 \times \omega_2)$. □

4.2 Nested model

In this subsection, we discuss the simulation of the nested model (e.g., [26, 30]) in the Tarski approach. For a brief introduction to the model, we refer to [13, 16], the formalism of which we shall adopt here.

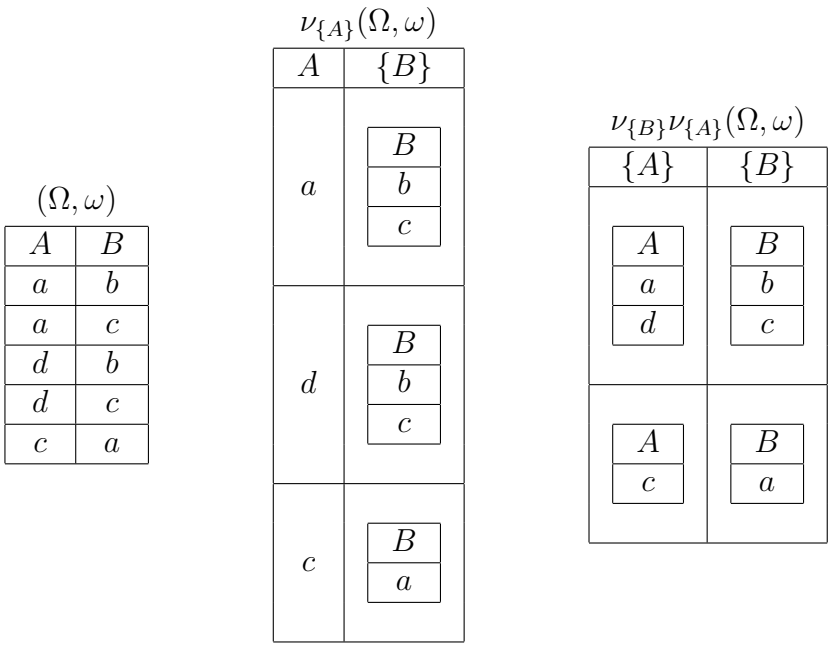


Figure 13: Nesting a relation

For the sake of concreteness and clarity, we shall use the very simple Codd relation (Ω, ω) shown in Figure 13 *(left)* as a running example. Figure 13 *(middle)* shows $\nu_{\{A\}}(\Omega, \omega)$, the result of nesting (Ω, ω) over $\{A\}$ and Figure 13 *(right)* shows $\nu_{\{B\}}\nu_{\{A\}}(\Omega, \omega)$, the result of nesting $\nu_{\{A\}}(\Omega, \omega)$ over $\{B\}$.

Ways of representing nested relations have already been discussed in the Introduction. We shall therefore concentrate on how these representations can be obtained in the Tarski algebra, starting for instance from the representation of a flat relation.

Therefore, let $\{r_A, r_B\}$ be some representation of (Ω, ω) in the Tarski model. By Theorem 4.1, we know that there exists a representation r'_{AB} of $\pi_{\{A\}}(\Omega, \omega)$, as shown in Figure 14 *(left)*. Figure 14 *(middle)* shows the relation $r'_B = r'_{AB} \cdot (r_A)^{-1} \cdot r_B$.

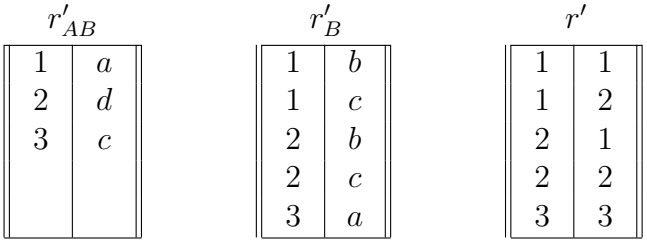


Figure 14: A representation of $\nu_{\{A\}}(\Omega, \omega)$

Alternatively, the tags in Figure 14 can be interpreted as representing the sets of B-values occurring in $\nu_{\{A\}}(\Omega, \omega)$. From this standpoint, the relation in Figure 14 *(left)* can be seen as

associating A-values with tags representing sets of B-values while the relation in Figure 14 *(middle)* identifies these sets.

Unfortunately, and contrary to the representation proposed in the Introduction, the same set of B-values can have more than one tag, as is the case in our example. Although we cannot do away with multiply represented sets in the Tarski algebra, we can compute all pairs of tags representing the same set. Turning to our example, let

$$\tilde{r}'_B = r'_B \cdot (r'_B)^{\text{un}} \cdot r'_B - r'_B$$

In $\tilde{r}'_B$, each tag value is associated to all values *not* belonging to the set of B-values identified by that tag. Using the set-theoretical property that two sets are different if and only if the first set and the complement of the second have a nonempty intersection, it follows that $r'_B \cdot (\tilde{r}'_B)^{-1}$ is the set of all pairs of tags representing different sets of B-values and that

$$r' = r'_B \cdot (r'_B)^{\text{un}} \cdot (r'_B)^{-1} - r'_B \cdot (\tilde{r}'_B)^{-1}$$

shown in Figure 14 *(right)* is the set of all pairs of tags representing the same set of B-values.

Hence $\{r'_{AB}, r'_B, r'\}$ can be seen as a representation of $\nu_{\{A\}}(\Omega, \omega)$.

r''_{AB}

1	1
2	2
3	3

r''_A

1	a
1	d
2	a
2	d
3	c

r''_B

1	b
1	c
2	b
2	c
3	a

r''

1	1
1	2
2	1
2	2
3	3

Figure 15: A representation of $\nu_{\{B\}}\nu_{\{A\}}(\Omega, \omega)$

Figure 15 shows a possible representation of $\nu_{\{B\}}\nu_{\{A\}}(\Omega, \omega)$. The tags in $r''_A = r' \cdot r'_{AB}$ can now also be seen as representing the sets of A-values occurring in $\nu_{\{B\}}\nu_{\{A\}}(\Omega, \omega)$. $r''_B = r'_B$ still shows the set of B-values in $\nu_{\{B\}}\nu_{\{A\}}(\Omega, \omega)$. $r''_{AB} = r'_A \cdot (r'_A)^{-1}$ associates sets of A-values and sets of B-values and, finally, $r'' = r'$ indicates which pairs of tags identify the same set of B-values (and hence also the same set of A-values).

To some extent, it is remarkable that the nested model can be simulated in the Tarski model without having to add complex constructs to the latter. However, this fact is not so surprising when seen in the perspective of the main result of [25]. There it has been shown that all nested algebra queries on flat relations returning a flat relation can be expressed in the flat relational algebra as well. In other words, the nest operator and its counterpart the unnest operator do not add anything essential to the language in terms of expressive power and only affect the way in which the data are grouped. In the Tarski algebra, this grouping is achieved using tagging.

4.3 GOOD model

In Section 2, we already explained how to represent a GOOD instance [14, 15] in the Tarski model. In particular, Figure 8 represents the GOOD instance shown in Figure 5. In this

subsection, we shall discuss how the three queries dealt with in Subsection 2.1 and shown in Figure 6 can be solved in the Tarski algebra. Due to space limitations, this discussion will be held rather informal. We trust the reader by now has acquired sufficient familiarity with the Tarski algebra to fill in the technical details.

The first query, shown in Figure 6 *(left)*, was a node addition adding a new node for each pair of parents having a common child. Using the representation in Figure 8, this query can be solved as follows in the Tarski algebra. First, the relation $ch \cdot ch^{-1}$ contains the set of all pairs of parent nodes having a common child. For these, new nodes can be "created" by tagging $ch \cdot ch^{-1}$. The left and right tagging automatically provide representations for the newly introduced edge labels "1" and "2", respectively. A suitable node label for the newly created nodes could be obtained from tagging the relation $(\mathit{nodelabels}^{-1} \cdot \mathit{nodelabels})^{\mathrm{id}}$ as the tag associated to the pair (P, P).

The second query, shown in Figure 6 *(middle)*, was an edge addition associating children directly to parent pairs. In the Tarski representation, a relation representing the edge label c must then be added. This can easily be achieved by composing the relations representing the edge labels "1" and "2" with ch, and taking their intersection.

Finally, the third query, shown in Figure 6 *(right)*, was an edge deletion removing all the original parent-children links. This operation can be simulated straightforwardly by removing the relation ch.

At this point, it should be emphasized that more elaborate GOOD queries, involving unbounded iteration, *cannot* be expressed in the Tarski algebra. In order to overcome this problem, we shall discuss ways in the next section to extend the Tarski algebra to a computationally complete language.

5 The extended Tarski algebra

From Theorem 4.1, in which we showed that the Tarski algebra can emulate Codd-relational queries, it follows that the Tarski algebra is at least as expressive as the standard (Codd-) relational algebra. It should be clear however that, as already alluded to at the end of the previous section, there remain queries which can not be expressed in the Tarski algebra. The classical example of such a query is the *transitive closure* of a binary relation. (If r is a binary relation, then its transitive closure is defined as the smallest relation s containing r, such that whenever (u, v) and (v, w) are in s, (u, w) is also in s.) The usual solution to this problem is to add a looping construct to the language. This will be done in Subsection 5.1. To examine the expressiveness of this *extended Tarski algebra*, we need to re-investigate the notion of "generic query". This will be the subject of Subsection 5.2. In Section 5.3 we will then show that the extended algebra is computationally complete for the class of generic queries defined in Section 5.2, thus emphasizing the combined power of tagging and unbounded looping in the context of mathematical relations.

5.1 While-expressions

In this subsection, we will introduce the extended Tarski algebra obtained by augmenting the Tarski algebra with assignment statements and while-loops. These additional constructs

will be written in a program-like fashion so as to make their semantics obvious.

In the previous section, we saw that in order to simulate a relation or a database in another model, several binary relations are needed. Therefore we shall introduce assignment queries in which several binary relations can be computed in one expression. These assignment queries will then turn out to be key constructs in the definition of an iterative query.

We define:

Definition 5.1 *Let $\vec{u}_1, \ldots, \vec{u}_l$ be finite sequences of relation variables. An* assignment query *$E(x_1, \ldots, x_m)$ is an expression of the form:*

query $E(x_1, \ldots, x_m)$;
  begin
    $z_1 := E_1(\vec{u}_1)$;
    $z_2 := E_2(\vec{u}_2)$;
      $\vdots$
    $z_l := E_l(\vec{u}_l)$;
    return $(y_1, \ldots, y_n)$
  end

where, for $j = 1, \ldots, l$,

1. *the variables in $\vec{u}_j$ belong to $\{x_1, \ldots, x_m, z_1, \ldots, z_{j-1}\}$;*
2. *z_j may belong to $\{x_1, \ldots, x_m, z_1, \ldots, z_{j-1}\}$; and*
3. *$E_j(\vec{u}_j)$ is an (extended) Tarski algebra expression returning a single relation,*

and $y_1, \ldots, y_n$ belong to $\{x_1, \ldots, x_m, z_1, \ldots, z_l\}$.

Given a relation r, we shall make no distinction between r and the 1-tuple (r). In this way, the query remains meaningful if the right-hand side of one of the assignments is itself, e.g., an assignment query returning one relation. Conversely, this assumption also allows us to interpret ordinary Tarski algebra expressions as special cases of assignment queries.

Using assignment queries, we can define iterative queries:

Definition 5.2 *Let $\vec{u}$ and $\vec{v}$ be finite sequences of variables, and let u_k be the last element of $\vec{u}$. An* iterative query *$E(x_1, \ldots, x_m)$ is an expression of the form:*

query $E(x_1, \ldots, x_m)$;
  begin
    $\vec{u} := F(x_1, \ldots, x_m)$;
    while $u_k \neq \emptyset$
      do
        $\vec{u} := G(\vec{v})$
      od;
    return $(y_1, \ldots, y_n)$
  end

where $y_1, \ldots, y_m$, and the variables in $\vec{v}$ are in $\vec{u}$ or in $\{x_1, \ldots, x_m\}$, and F and G are assignment queries.

Example 5.1 The following is an iterative query for the transitive closure query mentioned in the introduction to this section:

```
query E(x);
  begin
    y := x;
    z := y · y − y;
    while z ≠ ∅
      do
        y := y ∪ x · y;
        z := y · y − y;
      od;
    return y
  end
```

The smallest set of expressions containing the taes and closed under assignment and iteration will be called the *extended Tarski algebra*; we will denote its expressions as *etae*'s.

We observe that etaes may also contain *conditional statements*. These can indeed be replaced by a number of ordinary assignments as can be easily seen using an argument analogous to the proof of Proposition 3.5.

We claim that adding assignment and iteration to the Tarski algebra makes this query language computationally complete. In order to substantiate this claim, however, we need to review the notion of genericity in the presence of tags.

5.2 Generic queries

In this subsection we will discuss the notion of genericity in query environments wherein tags can be introduced. We would like to emphasize that the contents of this section are completely independent of any specific query language. For us, a *query* is merely a (partial) mapping from $\mathcal{R}^m$ to $\mathcal{R}^n$ for some m and n, where $\mathcal{R}$ is the set of all (mathematical) relations over $V = D \cup T$.

The notion of genericity was originally introduced by Aho and Ullman [3] and by Chandra and Harel [7] in the context of the relational model and has subsequently become a subject of frequent study in other query environments (e.g., [1, 2, 17]). The principle of all notions of genericity that are around is that a generic query should commute with all permutations on the set of data values. While this principle can immediately be applied to, e.g., the relational model, this is not the case for more elaborate database models where the result of a query may contain objects or values not present in one of the arguments. In these cases, the original permutation must first be extended in some "natural" way to the newly introduced objects or values.

Therefore we shall first discuss how a permutation ψ on the set of data values can naturally be extended to a permutation ψ^e on V. This will be done in an iterative way. Let $D_0 = D$, and let for all $i \geq 0$, $\mathcal{V}_i$ be the largest subset of $\mathcal{V}^2$ in which only values from

[2]Recall that $\mathcal{V}$ is the set of all pairs (s, t) with s a relation and t an ordered pair over V (Subsection 3.3).

D_i are used. Finally, let, for all $i \geq 1$, $D_i = D \cup \tau(\mathcal{V}_{i-1})$[3]. We then define ψ^e recursively: if $v \in D_0 = D$ then $\psi^e(v) = \psi(v)$, otherwise, if $v \in D_i$, $i \geq 1$, and $\tau^{-1}(v) = (s,t)$, then $\psi^e(v) = \tau\big(\psi^e(s), \psi^e(t)\big)$. Let $D_\infty = \bigcup_{i\geq 0} D_i$. Clearly D_∞ is a subset of V. We will make the assumption that D_∞ is equal to V, i.e., that all tag values in T are effectively used.

Lemma 5.1 *The function ψ^e has the following properties:*

1. *the restriction of ψ^e to D equals ψ,*
2. *the restriction of ψ^e to T equals $\tau\psi^e\tau^{-1}$, and*
3. *ψ^e is a permutation on V.*

Proof: Since Properties 1 and 2 follow directly from the definition of ψ^e, we immediately turn to Property 3. Since $V = \bigcup_{i\geq 0} D_i$ and $D_{i-1} \subseteq D_i$ for $i \geq 0$, it suffices to establish by induction that for each i, $i \geq 0$, the restriction of ψ^e to D_i is a permutation on D_i. Since $D_0 = D$, it follows from Property 1 that ψ^e is permutation on D_0. Assume that ψ^e is a permutation on D_i, $i \geq 0$. We need to prove that ψ^e is also a permutation on D_{i+1}.

First, we show that ψ^e is an injection. Therefore, let x_1, $x_2 \in D_{i+1}$ be such that $\psi^e(x_1) = \psi^e(x_2)$. If x_1 and x_2 are both in D_i, then $x_1 = x_2$, by the induction hypothesis. So assume that one of these two values, say x_1, is an element of $D_{i+1} - D_i$. Let $(s_1, t_1) \in \mathcal{V}_i$ be such that $\tau(s_1, t_1) = x_1$. Then by Property 2, $\psi^e(x_1) = \tau\big(\psi^e(s_1), \psi(t_1)\big)$. If x_2 would be in D_0 then $\psi^e(x_2) = \psi(x_2) = \tau\big(\psi^e(s_1), \psi(t_1)\big) = \psi^e(x_1)$, a contradiction, because $\psi(x_2)$ is in D. Hence for some k, $0 \leq k \leq i$, there exists (s_2, t_2) in $\mathcal{V}_k$ (and hence also in $\mathcal{V}_i$) such that $\tau(s_2, t_2) = x_2$. Then by Property 2, $\psi^e(x_2) = \tau\big(\psi^e(s_2), \psi(t_2)\big)$. Now, because $\psi^e(x_1) = \psi^e(x_2)$ and τ is a bijection, it follows that $\big(\psi^e(s_1), \psi^e(t_1)\big) = \big(\psi^e(s_2), \psi^e(t_1)\big)$. Since s_1, s_2, t_1 and t_2 take all their values in D_i, it follows by induction that $(s_1, t_1) = (s_2, t_2)$ and therefore $x_1 = x_2$. Thus we have established that ψ^e is an injection on D_{i+1}.

Let us now finally prove that ψ^e is also a surjection. Let $x \in D_{i+1}$. If $x \in D_i$, it follows from the induction hypothesis that x is in the range of ψ^e restricted to D_i and we are done. Therefore assume that $x \in D_{i+1} - D_i$ and let $(s,t) \in \mathcal{V}_i$ be such that $\tau(s,t) = x$. Since s and t take all their values in D_i, it follows from the induction hypothesis that there exist an s' and a t', which take their values in D_i, such that $\psi^e(s') = s$ and $\psi^e(t') = t$. By Property 2 it now easily follows that $x = \psi^e\big(\tau(\gamma', t')\big)$. Thus ψ^e is a also surjection on D_{i+1}, completing the proof. □

In Lemma 5.1 we established that any permutation ψ on D can be extended to a permutation ψ^e on V such that this extension *permutes* with the tagging function τ on T. In the next theorem we will establish that ψ^e is the *only* extension with this property.

Theorem 5.1 *Let ψ be a permutation on D. Then there exists a unique permutation ϕ on V which is an extension of ψ such that the restriction of ϕ to T equals $\tau\phi\tau^{-1}$.*

[3]All mappings under consideration are assumed to be extended to sets, ordered pairs and relations in the natural way.

Proof: The existence of the permutation ϕ follows from Lemma 5.1. We now show that there is only one such permutation.

Let ϕ_1 and ϕ_2 be permutations on V which are both extensions of ψ and which both permute with τ on T. Reconsider the sets D_i ($i \geq 0$). We will prove by induction that ϕ_1 and ϕ_2 agree on D_i for each $i \geq 0$. Since $V = \bigcup_{i \geq 0} D_i$ it will then follow that $\phi_1 = \phi_2$.

Since $D_0 = D$, it follows immediately that ϕ_1 and ϕ_2 agree on D_0. Assume by induction that ϕ_1 and ϕ_2 agree on D_i ($i \geq 0$). We need to prove that ϕ_1 and ϕ_2 also agree on D_{i+1}. Let therefore $x \in D_{i+1} - D_i$. Then there exists $(s,t) \in \mathcal{V}_i$ be such that $\tau(s,t) = x$. Since s and t take all their values in D_i, it follows from the induction hypothesis that $\phi_1(s,t) = \phi_2(s,t)$ and therefore also that $\tau\big(\phi_1(s,t)\big) = \tau\big(\phi_2(s,t)\big)$. Because both ϕ_1 and ϕ_2 permute with τ and because $x = \tau(s,t)$, it follows that $\phi_1(x) = \phi_2(x)$. Hence, ϕ_1 and ϕ_2 also agree on D_{i+1}. □

Let us denote by $\mathcal{S}_\tau$ the set of all permutations on V that permute with τ on T. Theorem 5.1 establishes that there is an injective mapping from the set of permutations over D into $\mathcal{S}_\tau$. This mapping sends a permutation ψ on D to its unique extension ψ^e on V which permutes with τ. In the next theorem, we will establish that this mapping is actually a bijection.

Theorem 5.2 *Let ϕ be a permutation over V which permutes with τ on T. Then there exists a permutation ψ over D such that $\psi^e = \phi$, where ψ^e is the unique extension of ψ the restriction of which to T permutes with τ.*

Proof: First we show that the restriction of ϕ to D must be a permutation on D.

Let $x \in D$ and consider $\phi(x)$. We need to show that $\phi(x)$ is in D. Assume that $\phi(x) \in T$ Since $T = \bigcup_{i \geq 0} D_i - D_0$, it follows that there exists a context s and a pair t taking their values in V such that $\phi(x) = \tau(s,t)$. Thus $x = \phi^{-1}\big(\tau(s,t)\big)$. Because the restriction of ϕ to T permutes with τ, it follows that also $x = \tau\big(\phi^{-1}(s), \phi^{-1}(t)\big)$ implying that $x \in T$, a contradiction.

Next we need to show that if $x \in T$ then $\phi(x)$ is also in T. Since $x \in T$, we can again find a context s' and a pair t' taking their values in V such that $x = \tau(s',t')$. Thus $\phi(x) = \phi\big(\tau(s',t')\big) = \tau\big(\phi(s'), \phi(t')\big)$ is also in T.

Having established that the restriction to D of ϕ is a permutation on D, let ψ be this restriction. The theorem then immediately follows from Theorem 5.1. □

Theorem 5.2 states that there is natural bijection between the set of permutations over D and the set $\mathcal{S}_\tau$ of permutations over V that permute with τ on T. This allows us to finally define in a natural way the concept of a τ-generic query in the context of tagging.

Definition 5.3 *A query $Q : \mathcal{R}^m \to \mathcal{R}^n$ is* τ-generic *if for each $\phi \in \mathcal{S}_\tau$ and for each sequence of relations $r_1, \ldots, r_m$ in $\mathcal{R}$ such that $Q(r_1, \ldots, r_m)$ is defined one has that*

$$\phi\big(Q(r_1, \ldots, r_m)\big) = Q\big(\phi(r_1, \ldots, r_m)\big)$$

In the following proposition we establish that all the operators of the Tarski algebra define τ-generic queries.

Proposition 5.1 *The union, composition, inverse, complement, left-tagging, and right-tagging operators define τ-generic queries.*

Proof: We will only prove this proposition for the left-tagging operator. Let r and s be relations and let $\phi \in \mathcal{S}_\tau$. We have to establish that

$$\phi(r_s^{\triangleleft}) = \phi(r)_{\phi(s)}^{\triangleleft}$$

Let $t \in \phi(r_s^{\triangleleft})$. Then there exists a pair $(v, w) \in r$ such that

$$t = \Big(\phi\big(\tau(s, (v, w))\big), \phi(v)\Big)$$

Since the restriction of ϕ to T permutes with τ,

$$t = \Big(\tau\big(\phi(s), \phi(v, w)\big), \phi(v)\Big)$$

Thus $t \in \phi(r)_{\phi(s)}^{\triangleleft}$. A similar argument can be made to show the converse. □

We can now state our first main result.

Theorem 5.3 *If E is an extended Tarski expression then E defines a τ-generic query.*

Proof: This theorem follows by induction from Proposition 5.1. □

5.3 Computational completeness of the extended Tarski algebra

In this section we will establish that the extended Tarski algebra (relative to a fixed computable tagging function τ) is a computationally complete language for a large class of τ-generic queries.

In the previous subsection, we "decomposed" the set V of all values into a hierarchy of layers D_i. In order to define the class of τ-generic queries for which we will show the completeness of the extended Tarski algebra, we need to elaborate a little further on this decomposition into layers.

Definition 5.4 *Let τ be a tagging function. Let $W \subseteq V$. We define recursively:*

1. $W_0 = W$,
2. $\mathcal{W}_i$, $i \geq 0$, *is the largest subset of $\mathcal{V}$ in which only values from W_i are used; and*
3. *for $i > 0$, $W_i = W \cup \tau(\mathcal{W}_{i-1})$.*

The set $W_\infty = \bigcup_{i \geq 0} W_i$ is called the natural extension *of W (with respect to τ).*

We now define:

Definition 5.5 *Let τ be a tagging function. A query $Q : \mathcal{R}^m \to \mathcal{R}^n$ is called* τ-domain-preserving *if for all relations $r_1, \ldots, r_m$,*

$$Q(r_1, \ldots, r_m)\downarrow \subseteq (r_1\downarrow \cup \ldots \cup r_m\downarrow)_\infty$$

with $(r_1\downarrow \cup \ldots \cup r_m\downarrow)_\infty$ the natural extension of $r_1\downarrow \cup \ldots \cup r_m\downarrow$ with respect to τ.

The notion of *domain-preserving* originates from the fact that the (extended) Tarski algebra only allows for tagging, and not for "de-tagging". A query such as "find the minimal set of data values necessary to generate all values of a given input relation" is clearly beyond the power of the Tarski operators and must therefore be excluded. Obviously, we have:

Theorem 5.4 *If E is an extended Tarski expression then E defines a τ-domain-preserving query.*

We now have the following theorem.

Theorem 5.5 *Let τ be a computable tagging function. The extended Tarski algebra with tagging function τ is complete for the class of computable, τ-domain-preserving, τ-generic queries.*

Due to space limitations, we only give a sketch of the proof of Theorem 5.5. By Theorems 5.3 and 5.4, each extended Tarski expression defines a τ-domain-preserving, τ-generic query which is obviously computable. So we still need to prove that if Q is a computable, τ-domain-preserving, τ-generic query, then there exists an extended Tarski expression E_Q defining the same function.

Our proof follows that of a theorem by Abiteboul and Vianu ([2], Theorem 3.2.2, p. 212). This theorem states that the language *detTL* is complete with respect to the class of *typed* computable generic queries on Codd-relational databases. The key ideas of their proof are the following ([2], p. 202):

1. show that *detTL* can simulate counters and thus has full Turing machine capability, and

2. show that *detTL* can be used to compute the "Gödel number" of a Codd-relational database and, conversely, that a Codd-relational database can be computed in *detTL* from its Gödel number.

The computation then proceeds as follows. Let I be the input database and let $I\downarrow$ denote the set of domain elements in I. Consider the set $\Sigma(I)$ of *all* total orderings on $I\downarrow$. (Abiteboul and Vianu show that Σ can be computed in *detTL* and represented in a ternary relation.) Then compute (in parallel) for each order $\sigma \in \Sigma(I)$ the Gödel number $\Gamma(\sigma, I)$ of the input database I. The genericity of this auxiliary query guarantees that if σ_1 and σ_2 are two orderings on $I\downarrow$, then $\Gamma(\sigma_1, I)$ and $\Gamma(\sigma_2, I)$ represent the same number *relative to the respective orderings*. (In other words, the Gödel number of I is independent of the choice of a particular order on its domain $I\downarrow$.) Next, the simulation of a Turing machine is performed on the Gödel numbers of I yielding the Gödel numbers of the output database of the query. Due

to the genericity of the simulated query, the Gödel numbers of the resulting Codd-relational database still represent the same number when seen relative to the corresponding orderings. This property allows to decode the Gödel numbers unambiguously to finally yield the actual output database.

We can use the Abiteboul-Vianu proof for Theorem 5.5 if we can show that in the extended Tarski algebra:

1. for an input (mathematical) relational database I, the set $\Sigma(I)$ of all orderings over $I\downarrow$ can be computed, and

2. counter machines can be simulated.

Turning to the first item, it should first be mentioned that this claim too would follow from the Abiteboul-Vianu proof if it would only have been the case that the set $\Sigma(I)$ was represented by binary Codd relations. Unfortunately for us, Abiteboul and Vianu coded $\Sigma(I)$ as a *ternary* relation. To be more specific, let I be a database such that $I\downarrow = \{a, b, c\}$. Then Abiteboul and Vianu represent $\Sigma(I)$ as in Figure 16.

Σ	$<$	$>$
σ_1	♡	a
σ_1	a	b
σ_1	b	c
σ_1	c	♠
σ_2	♡	a
σ_2	a	c
σ_2	c	b
σ_2	b	♠
σ_3	♡	b
σ_3	b	a
σ_3	a	c
σ_3	c	♠
σ_4	♡	b
σ_4	b	c
σ_4	c	a
σ_4	a	♠
σ_5	♡	c
σ_5	c	a
σ_5	a	b
σ_5	b	♠
σ_6	♡	c
σ_6	c	b
σ_6	b	a
σ_6	a	♠

Figure 16: The ternary relation $\Sigma(I)$

In the ternary relation of Figure 16, the leftmost column (Σ) gives the names of the six total orderings on $\{a, b, c\}$. Then, for $1 \leq i \leq 6$ and $x, y \in \{a, b, c\}$,

1. a tuple of the form $(\sigma_i, \heartsuit, v)$ indicates that v is the smallest element in the ordering σ_i,
2. a tuple of the form $(\sigma_i, w, \spadesuit)$ indicates that w is the largest element in the ordering σ_i, and
3. a tuple of the form (σ_i, v, w) indicates that v is smaller than w in the ordering σ_i.

r_Σ

1	σ_1
2	σ_1
3	σ_1
4	σ_1
5	σ_2
6	σ_2
7	σ_2
8	σ_2
9	σ_3
10	σ_3
11	σ_3
12	σ_3
13	σ_4
14	σ_4
15	σ_4
16	σ_4
17	σ_5
18	σ_5
19	σ_5
20	σ_5
21	σ_6
22	σ_6
23	σ_6
24	σ_6

$r_<$

1	$\heartsuit$
2	a
3	b
4	c
5	$\heartsuit$
6	a
7	c
8	b
9	$\heartsuit$
10	b
11	a
12	c
13	$\heartsuit$
14	b
15	c
16	a
17	$\heartsuit$
18	c
19	a
20	b
21	$\heartsuit$
22	c
23	b
24	a

$r_>$

1	a
2	b
3	c
4	$\spadesuit$
5	a
6	c
7	b
8	$\spadesuit$
9	b
10	a
11	c
12	$\spadesuit$
13	b
14	c
15	a
16	$\spadesuit$
17	c
18	a
19	b
20	$\spadesuit$
21	c
22	b
23	a
24	$\spadesuit$

Figure 17: The three binary ordering relations r_Σ, $r_<$, and $r_>$

Obviously, our strategy is to show that this ternary relation can be computed in three binary relations. Thereto, we can simply employ the representation for a Codd relation outlined in Definition 4.1. In other words, our computation should represent the relation $\Sigma(I)$ in the three binary relations shown in Figure 17. In this figure, the numbers represent tags for the 24 tuples in $\Sigma(I)$, the relation r_Σ specifies which tuples belong to which ordering, the relation $r_<$ codes the "smaller sides" of the orderings, and the relation $r_>$ codes the "larger sides".

Notice that in this representation there are the two special constants $\heartsuit$ and $\spadesuit$. These constants are not strictly necessary. In our construction of $\Sigma(I)$, we will in fact dispense with the two special constants and will code a tuple of the form $(\sigma_i, v, \spadesuit)$ by the tuple (σ_i, v, σ_i) and a tuple of the form $(\sigma_i, \heartsuit, w)$ by the tuple (σ_i, σ_i, w). Although this decision makes the construction slightly more complicated, it has the benefit of being conceptually cleaner.

r_Σ

1	σ_1
2	σ_1
3	σ_1
4	σ_1
5	σ_2
6	σ_2
7	σ_2
8	σ_2
9	σ_3
10	σ_3
11	σ_3
12	σ_3
13	σ_4
14	σ_4
15	σ_4
16	σ_4
17	σ_5
18	σ_5
19	σ_5
20	σ_5
21	σ_6
22	σ_6
23	σ_6
24	σ_6

$s_<$

1	σ_1
2	a
3	b
4	c
5	σ_2
6	a
7	c
8	b
9	σ_3
10	b
11	a
12	c
13	σ_4
14	b
15	c
16	a
17	σ_5
18	c
19	a
20	b
21	σ_6
22	c
23	b
24	a

$s_>$

1	a
2	b
3	c
4	σ_1
5	a
6	c
7	b
8	σ_2
9	b
10	a
11	c
12	σ_3
13	b
14	c
15	a
16	σ_4
17	c
18	a
19	b
20	σ_5
21	c
22	b
23	a
24	σ_6

Figure 18: The three binary ordering relations r_Σ, $s_<$, and $s_<$, no longer containing special constants

Before we proceed to show how to compute $\Sigma(I)$, let us now turn to our second task and show how counter machines can be simulated in the extended Tarski algebra once $\Sigma(I)$ has been computed.

A counter will be represented by a (binary) relation. We will assume that if the contents of this binary relation is empty, then the counter has the value "0". Otherwise its value, say "j", $j > 0$, will be represented by the relation $\{(\sigma_i(j), \sigma_i) \mid \sigma_i \in \Sigma(I)\}$, where $\sigma_i(j)$ denotes the jth element in the ordering σ_i. This representation will be derived from the binary relations representing $\Sigma(I)$. For example, according to Figure 18, "1" is represented by the relation

$$\{(a, \sigma_1), (a, \sigma_2), (b, \sigma_3), (b, \sigma_4), (c, \sigma_5), (c, \sigma_6)\}$$

"2" is represented by the relation

$$\{(b, \sigma_1), (c, \sigma_2), (a, \sigma_3), (c, \sigma_4), (a, \sigma_5), (b, \sigma_6)\}$$

and "3" is represented by the relation

$$\{(c, \sigma_1), (b, \sigma_2), (c, \sigma_3), (a, \sigma_4), (b, \sigma_5), (a, \sigma_6)\}$$

How can we represent "4"? To do this, we need to expand each of the orderings σ_i, $1 \leq i \leq 6$, with one new "largest" element, using tagging. Thereto, consider the tae

$$\mathrm{eoo}(x_\Sigma, y_<, y_>) \equiv x_\Sigma \cap y_>$$

Clearly, $\mathrm{eoo}(r_\Sigma, s_<, s_>) = r_\Sigma \cap s_>$ contains the largest element of each ordering ("end of ordering"). Let q be an arbitrary relation. The tags introduced in $\mathrm{eoo}(r_\Sigma, s_<, s_>)^{\triangleleft}_q$ and $\mathrm{eoo}(r_\Sigma, s_<, s_>)^{\triangleright}_q$ can serve as new "largest" elements for each of the orderings, their novelty following from the injectivity of the tagging function τ. In order to show that the binary relations r_Σ, $s_<$ and $s_>$ can be modified accordingly, consider the following assignment query.

query expand$(x_\Sigma, y_<, y_>, u)$;
  begin
    $x_\Sigma := x_\Sigma \cup \mathrm{eoo}(x_\Sigma, y_<, y_>)^{\triangleright}_u$;
    $y_< := y_< \cup \mathrm{eoo}(x_\Sigma, y_<, y_>)^{\triangleleft}_u \cdot \big(\mathrm{eoo}(x_\Sigma, y_<, y_>)^{\triangleleft}_u\big)^{-1}$;
    $y_> := \big(y_> - \mathrm{eoo}(x_\Sigma, y_<, y_>)\big) \cup \mathrm{eoo}(x_\Sigma, y_<, y_>)^{\triangleright}_u$
      $\cup\ \mathrm{eoo}(x_\Sigma, y_<, y_>) \cdot \big(\mathrm{eoo}(x_\Sigma, y_<, y_>)^{\triangleright}_u\big)^{-1}$;
    return $(x_\Sigma, y_<, y_>)$
  end

Let $(r'_\Sigma, s'_<, s'_>) = \mathrm{expand}(r_\Sigma, s_<, s_>, q)$. Obviously, the binary relations r'_Σ, $s'_<$ and $s'_>$, shown in Figure 19, are the correct expansions of r_Σ, $s_<$ and $s_>$, respectively.

The number "4" is now represented by the relation

$$\{(25, \sigma_1), (26, \sigma_2), (27, \sigma_3), (28, \sigma_4), (29, \sigma_5), (30, \sigma_6)\}$$

i.e., by the newly introduced tag values as "largest" values of their respective orderings.

The above technique shows that we can always expand the relations r_Σ, $s_<$, and $s_>$ to represent an arbitrary large natural number. However, there is a small caveat. If I is the

r'_Σ	
1	σ_1
2	σ_1
3	σ_1
4	σ_1
5	σ_2
6	σ_2
7	σ_2
8	σ_2
9	σ_3
10	σ_3
11	σ_3
12	σ_3
13	σ_4
14	σ_4
15	σ_4
16	σ_4
17	σ_5
18	σ_5
19	σ_5
20	σ_5
21	σ_6
22	σ_6
23	σ_6
24	σ_6
25	σ_1
26	σ_2
27	σ_3
28	σ_4
29	σ_5
30	σ_6

$s'_<$	
1	σ_1
2	a
3	b
4	c
5	σ_2
6	a
7	c
8	b
9	σ_3
10	b
11	a
12	c
13	σ_4
14	b
15	c
16	a
17	σ_5
18	c
19	a
20	b
21	σ_6
22	c
23	b
24	a
25	25
26	26
27	27
28	28
29	29
30	30

$s'_>$	
1	a
2	b
3	c
4	25
5	a
6	c
7	b
8	26
9	b
10	a
11	c
12	27
13	b
14	c
15	a
16	28
17	c
18	a
19	b
20	29
21	c
22	b
23	a
24	30
25	σ_1
26	σ_2
27	σ_3
28	σ_4
29	σ_5
30	σ_6

Figure 19: An expansion of the relations r_Σ, $s_<$ and $s_>$

empty database, then we need to guarantee that the three binary relations representing $\Sigma(I)$ are all nonempty. Otherwise, it is impossible to introduce new values by tagging. This can be accommodated by assuming that there is one special constant, say "♣", and that r_Σ, $s_<$, and $s_>$ are initialized to the singleton $\{(\clubsuit, \clubsuit)\}$.[4]

To prove that the extended Tarski algebra can simulate counter machines we need to show that

1. the contents of a counter can be incremented by one,
2. the contents of a counter with a strictly positive value can be decremented by one, and
3. a while-loop program, which loops until a control counter becomes zero, can be simulated.

Assume that counter *ctr-k* has value j. As counters are constructed from the three binary relations r_Σ, $s_<$ and $s_>$, we may assume that these relations already allow the representation of j. In order to increment *ctr-k* by one, we first need to check whether or not j is the *largest* number of which r_Σ, $s_<$ and $s_>$ allow the representation. If this is the case, then we first have to expand r_Σ, $s_<$ and $s_>$ before we can actually increment the counter. This results in the following extended Tarski algebra expression:

query increment$(x_\Sigma, y_<, y_>, z_k, u)$;
  begin
    if $z_k - \mathrm{eoo}(x_\Sigma, y_<, y_>) = \emptyset$
      then
        $(x_\Sigma, y_<, y_>) := \mathrm{expand}(x_\Sigma, y_<, y_>, u)$;
    $z_k := y_>^{-1} \cdot (x_\Sigma \cap y_< \cdot z_k)$;
    return $(x_\Sigma, y_<, y_>, z_k)$)
  end

If q is an arbitrary relation and if $(r'_\Sigma, s'_<, s'_>, \textit{ctr-k}') = \mathrm{increment}(r_\Sigma, s_<, s_>, \textit{ctr-k}, q)$, then the value of *ctr-k*′ is the value of *ctr-k* incremented by one. Furthermore, we have either $(r'_\Sigma, s'_<, s'_>) = (r_\Sigma, s_<, s_>)$ or $(r'_\Sigma, s'_<, s'_>) = \mathrm{expand}(r_\Sigma, s_<, s_>, q)$, depending on whether or not an expansion was necessary.

Similarly, it is possible to write a Tarski expression to decrement a counter. Finally since the extended Tarski algebra has while loops, we can also meet Condition 3 in the list of requirements to simulate counter machines.

So, all we have left to show is that it is possible to write an extended Tarski expression to compute the initial contents of r_Σ, $s_<$, and $s_>$, as, e.g., in Figure 18 for a database I with domain $\{a, b, c\}$.

The first step in this computation is to initialize the relations r_Σ, $s_<$, and $s_>$ and to compute the domain *dom*. Let us assume that the input database I consists of the binary

[4]This assumption requires that we slightly adjust the notion of τ-genericity by defining it only relative to the permutations on D which keep ♣ fixed.

relations $(r_1, \ldots, r_m)$ and consider the assignment query:

query initialize$(x_1, \ldots, x_m)$;
  begin
    $x_\Sigma := \{(\clubsuit, \clubsuit)\}$;
    $y_< := \{(\clubsuit, \clubsuit)\}$;
    $y_> := \{(\clubsuit, \clubsuit)\}$;
    $x_{dom} := x_1^{\mathrm{id}} \cup \ldots \cup x_m^{\mathrm{id}}$;
    return $(x_\Sigma, y_<, y_>, x_{dom})$
  end

The initialization is realized by putting $(r_\Sigma, s_<, s_>, dom) = \text{initialize}(r_1, \ldots, r_m)$.

Next, one incrementally builds the (partial) orders of size $1, 2, \ldots$, up to the orders of size $|dom|$ (i.e., the complete orders). In our example, where $dom = \{(a,a),(b,b),(c,c)\}$, one builds in the first iteration the orders of size one, i.e., (a), (b), and (c), in the second iteration the orders of size two, i.e., (a,b), (a,c), (b,a), (b,c), (c,a), and (c,b), and finally, in the third iteration, the orders of size three, i.e., (a,b,c), (a,c,b), (b,a,c), (b,c,a), (c,a,b), and (c,b,a). So the algorithm has the following structure:

query build-up$(x_1, \ldots, x_m, u_1, u_2)$;
  begin
    $(x_\Sigma, y_<, y_>, x_{dom}) := \text{initialize}(x_1, \ldots, x_m)$;
    $notused := x_\Sigma \cdot \overline{x_\Sigma \cup x_{dom}} \cdot x_{dom}$;
    while $notused \neq \emptyset$
      do
        $min := x_\Sigma \cap y_<$;
        $max := x_\Sigma \cap y_>$;
        $temp := notused^{\triangleleft}_{u_1} \cdot x_\Sigma^{-1} \cup notused^{\triangleright}_{u_1}$;
        $(x_\Sigma, y_<, y_>) := \text{extend}(x_\Sigma, y_<, y_>, x_{dom}, min, max, temp, u_2)$;
        $used := x_\Sigma^{-1} \cdot y_< \cdot x_{dom}$;
        $notused := \overline{used} \cdot x_{dom}$;
      od;
    return $(x_\Sigma, y_<, y_>)$
  end

In *build-up*, *notused* is initialized to $\{(\clubsuit, v) \mid v \in I{\downarrow}\}$ to indicate that no value has yet been used in any ordering. Inside the while-loop, the query *extend* transforms the existing partial orders to partial orders of size one larger, using the values of *min*, *max* and *temp*.[5] To the variable *used*, the relation is assigned consisting of all pairs (σ, v) with σ an existing partial order and v a domain value occurring in that order. Finally, *notused* will consist of all pairs (σ, w) with σ an existing partial order and w a domain value *not* occurring in σ.

What remains to be explained is how *extend* is written. To follow the appropriate computations, it is useful to first consider Figure 20.

In Figure 20, the triples of relations $(r_\Sigma^i, s_<^i, s_>^i)$, $0 \leq i \leq 2$, are the states of the relations r_Σ, $s_<$, and $s_>$ just before the $(i+1)$th iteration of the main while loop in *build-up*.

[5] Observe in particular that *min* and *max* compute the "first" respectively the "last" tuple of each ordering.

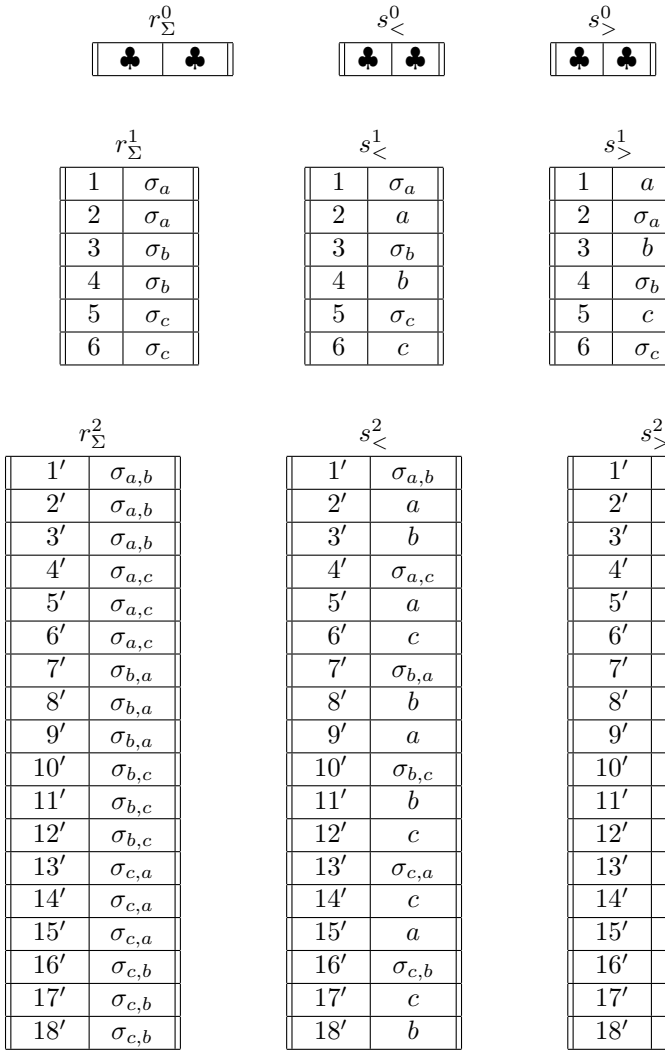


Figure 20: The states of the relations r_Σ, $s_<$, and $s_>$ just before the first, second and third iterations of *build-up*

The first step in the extension involves the computation of the tags for the new partial orderings. Remember that the variable *notused* lists for each current order, say σ, the domain elements *not* used in ω. Each such element therefore determines a different extension of σ. Therefore, obtaining tags for the extended orderings can be achieved by tagging *notused*.

Once the tags are obtained for the new (partial) orderings, it is possible to compute the new values for r_Σ, $s_<$ and $s_>$. We first consider r_Σ:

query new-$\Sigma(x_\Sigma, temp, u_2)$;
  begin
    $x_\Sigma := temp^{\triangleleft}_{u_2}$;
    return x_Σ
  end

Computing the new values for $s_<$ and $s_>$ is more involved. In order to write down the corresponding etaes more easily, we define for a relation r the *left identity operator*:

$$r^{\mathrm{idl}} = \{(v, v) \mid \exists w \in V : (v, w) \in r\}$$

Since $r^{\mathrm{idl}} = (r^{\triangleleft}_s)^{-1} \cdot r^{\triangleleft}_s$ with s an arbitrary relation, the left identity operator can be expressed in the Tarski algebra. The expression computing the new value for $s_<$ is:

query new-$<(x_\Sigma, y_<, x_{dom}, min, temp, u_2)$;
  begin
    $sm\text{-}first := (temp^{\triangleright}_{u_2} \cdot min^{\mathrm{idl}})^{\mathrm{idl}} \cdot x_\Sigma$;
    $sm\text{-}last := temp^{\triangleright}_{u_2} \cdot x_{dom}$;
    $sm\text{-}temp := x^{\mathrm{idl}}_\Sigma - (sm\text{-}first \cup sm\text{-}last)^{\mathrm{idl}}$;
    $sm\text{-}rest := sm\text{-}temp \cdot temp^{\triangleright}_{u_2} \cdot y_<$;
    $y_< := sm\text{-}first \cup sm\text{-}last \cup sm\text{-}rest$;
    return $y_<$
  end

The new value of $s_<$ is obtained in three parts: *sm-first* contains the "first" pair of each ordering, *sm-last* contains the "last" pair of each ordering and *sm-rest* is used to compute the remaining pair. Similarly, the new value of $s_>$ is also computed in three parts:

query new-$>(x_\Sigma, y_>, x_{dom}, max, temp, u_2)$;
  begin
    $gr\text{-}last := (temp^{\triangleright}_{u_2} \cdot x_{dom})^{\mathrm{idl}} \cdot x_\Sigma$;
    $gr\text{-}onebutlast := (temp^{\triangleright}_{u_2} \cdot max^{\mathrm{idl}})^{\mathrm{idl}} \cdot x_\Sigma \cdot temp$;
    $gr\text{-}temp := x^{\mathrm{idl}}_\Sigma - (gr\text{-}last \cup gr\text{-}onebutlast)^{\mathrm{idl}}$;
    $gr\text{-}rest := gr\text{-}temp \cdot temp^{\triangleright}_{u_2} \cdot y_>$;
    $y_> := gr\text{-}last \cup gr\text{-}onebutlast \cup gr\text{-}rest$;
    return $y_>$
  end

Using the etaes introduced above, we can write the following assignment query for *expand*:

query extend$(x_\Sigma, y_<, y_>, x_{dom}, min, max, temp, u_2)$;
  begin
    $x_\Sigma :=$ new-$\Sigma(x_\Sigma, temp, u_2)$;
    $y_< :=$ new-$<(x_\Sigma, y_<, min, temp, u_2)$;
    $y_> :=$ new-$>(x_\Sigma, y_>, max, temp, u_2)$;
    return $(x_\Sigma, y_<, y_>)$
  end

which concludes this sketch of the completeness proof.

We want to end this section with two final remarks concerning the proof above. Firstly, it should be mentioned that only the contexts explicitly used in the etaes above were mentioned in the parameter lists of these etaes. The use of the identity and left identity operators implicitly involves contexts too. Secondly, *all* contexts—whether explicit or implicit—can be seen as parameters of the final etae E_Q simulating Q and must not be computed from other data. As a consequence, each of three tagging strategies discussed in Subsection 3.3 (tuple tagging, relation tagging and universal tagging) already suffices to guarantee completeness.

6 Directions for future research

We conclude this article with a (non-exhaustive) list of topics not addressed in this paper, but clearly relevant for tagged-based database models in general and for the Tarski database model in particular. Specifically, we will discuss:

- *tagging of complex objects* other than ordered pairs;
- *constraints* such as functional dependencies in the Tarski database model; and
- *implementation considerations* for the Tarski database model and the Tarski algebra.

Turning to the first item, it should be emphasized that in this article, we limited ourselves (deliberately) to the use of tagging functions which associate tag values to *ordered pairs*. In Subsection 4.2 we showed that, even within this limited setting, it is possible to interpret certain tags as representations for finite sets. We also indicated, however, that, unlike for ordered pairs, we cannot guarantee that a finite set of objects is represented by a *unique* tag. Thus we are confronted with the problem of multiple tags (copies) for the same complex object. A simple way to overcome this problem is to admit tagging functions defined over domains of *complex objects other than ordered pairs*. We could for example consider a tagging function associating a unique tag to each finite set of values (i.e., data values or tag values). Such a tagging function would allow the definition of *finite-set tagging operators* in addition to the (ordered-pair) tagging operators already considered in this paper. This approach would of course overcome the multiple-copies problem and would yield a cleaner language to simulate the nested relational algebra. It should however be emphasized that we would still remain within the realm of binary (flat) relations.

An important database design tool for relational database we would like to discuss here in the context of tagging is the theory of *constraints*. The best-known examples of such constraints for Codd-relational databases are functional dependencies. Clearly, it is reasonable to assume that such constraints play as pivotal a role in Tarski-relational databases. The fact we wish to elicit here is that it is possible to use Tarski algebra expressions to formulate such constraints, as was already pointed out by Tarski himself in his original paper on the calculus of (binary) relations [28].

The underlying idea is quite simple. Suppose that we have a binary relation f. In general f will not define a function. However, if we postulate that f satisfy the constraint

$$f^{-1} \cdot f \ \subseteq \ f^{id}$$

then f clearly defines a function. This observation is the key to the problem of formulating (general) functional dependencies.

Of course, other types of constraints can equally well be expressed in this way. Suppose for example that we wish to state that r is a transitively closed binary relation. This can be formulated with the constraint $r \cdot r \subseteq r$.

We wish to conclude this article with some observations concerning the (possible) implementation of the Tarski database model and the Tarski algebra. As it turns out, various researchers have proposed storage architectures to facilitate such implementations. We are thinking in particular about the so called *decomposed storage model* [12, 21]. In the decomposed storage model, the database is internally stored as a collection of binary relations. So-called *surrogates* (our tags) are used to conceptually connect arbitrary relations. Thus, in a strong sense, the data is highly partitioned.

As discussed convincingly in [12], however, this storage model offers many advantages. The most striking characteristic is the simplicity and uniformity of the data structures needed to store and access data. This leads in turn to simpler access data methods. In particular, the set of access operations is small and straightforward to implement. On the other hand, due to the fragmentation of the data, access methods typically translate in complex sequences of access operations. The developers of the decomposed storage model justifiably make the analogy to RISC technology for computer hardware. It is in such a sense that we think Tarski algebra expressions can best be viewed. Although each individual operation in such an expression is simple, the complexity resides in the entire expression.

The authors of [21] also hint at the potential to support the decomposed storage structure on parallel machines. In summary, there appears evidence in the literature of database systems that the Tarski database model can be effectively and efficiently supported. In an age wherein it is believed that the majority of improvements in performance will come through parallel computation, the Tarski database model presents itself as a serious competitor to the relational database model as a conceptual model to improve the performance of general purpose database management systems.

Acknowledgments

The authors would like to express their gratitude to the following persons: Ed Robertson for discussions about the Tarski approach; Vijay Sarathy and Ed for discussions on the

GOOD→Tarski mapping; George Springer for letting us use his "Scheme" programs to manipulate binary relations, which helped us in finding several counterexamples and allowed us to test certain Tarski programs; and Jan Van den Bussche for discussions about the genericity concept. All these persons greatly helped us in clarifying several issues discussed in this paper.

The first author also wishes to acknowledge the financial support of the Belgian National Fund for Scientific Research (NFWO) which enabled him to visit Indiana University, where most of this work was carried out.

References

[1] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Portland, Oregon, 1989, pp. 159–173.

[2] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences* 41, 1990, pp. 181–229.

[3] A.V. Aho and J.D. Ullman. Universality of data retrieval languages. In *Proc. 6th ACM SIGPLAN-SIGACT Symp. on Princ. Programming Languages*, San Antonio, Texas, 1979, pp. 110–117.

[4] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Proc. 1st Int'l Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, 1989, pp. 40–57.

[5] F. Bancilhon. Object-oriented database systems. In *Proc. 7th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. Database Systems*, Austin, Texas, 1988, pp. 152–162.

[6] M.J. Carey, editor. Special issue on extensible database systems. *Database Engineering*, 9:2, June 1987.

[7] A.K. Chandra and D. Harel. Computable queries for relational databases. *Journal of Computer and System Sciences*, 21, 1980, pp. 156–178.

[8] W. Chen and D. Warren. C-Logic of complex objects. In *Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. Database Systems*, Portland, Oregon, 1989, pp. 369–378.

[9] E.F. Codd. A relational model for large shared data banks. *Comm. of the ACM*, 13, 1970, pp. 377–387.

[10] E.F. Codd. Further normalizations of the relational data base model. In Data Base Systems, R. Rustin, Ed., Prentice Hall, Englewood Cliffs, N.J., 1972, pp. 33-64.

[11] E.F. Codd. Relational completeness of database sublanguages. In Data Base Systems, R. Rustin, Ed., Prentice Hall, Englewood Cliffs, N.J., 1972, pp. 65–98.

[12] G. Copeland and S. Khoshafian A decomposition storage model. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Austin, Texas, 1985, pp. 268–279.

[13] M. Gyssens, J. Paredaens, and D. Van Gucht. A uniform approach towards handling atomic and structured information in the nested relational database model. *Journal of the ACM*, 36:4, 1989, pp. 790–825.

[14] M. Gyssens, J. Paredaens, and Dirk Van Gucht. A graph-oriented object database model. In *Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. Database Systems*, Nashville, Tenn., 1990, pp. 417–424.

[15] M. Gyssens, J. Paredaens, and Dirk Van Gucht. A graph-oriented object model for database end-user interfaces. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Atlantic City, New Jersey, 1990, pp. 24–33.

[16] M. Gyssens and D. Van Gucht. A comparison between algebraic query languages for flat and nested databases. *Theoretical Computer Science*, 87, 1991, pp. 263–286.

[17] R. Hull and J. Su. Untyped sets, invention, and computable queries. In *Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. Database Systems*, Philadelphia, Penn., 1989, pp. 347–359.

[18] R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proc. 16th Int'l Conf. on Very Large Databases*, Brisbane, Australia, 1990.

[19] M. Kifer and G. Lausen. F-logic, a higher-order language for reasoning about objects, inheritance, and scheme. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Portland, Oregon, 1989, pp. 134–146.

[20] W. Kim and F.H. Lochovsky, editors. *Object-oriented concepts, databases, and applications*. ACM Press (Frontier Series), 1989.

[21] S. Khoshafian, G. Copeland, T. Jagodits, H. Boral and P. Valduriez. A query processing strategy for the decomposed storage model. In *Proc. 3rd IEEE Int'l Conf. on Data Engineering*, Los Angeles, Cal., 1987, pp. 636–643.

[22] K-D. Kraegeloh and P.C. Lockemann. Hierarchies of data base languages: an example. *Information Systems*, 1, 1975, pp. 79–90.

[23] G.M. Kuper and M.Y. Vardi. A new approach to database logic. In *Proc. 3rd ACM SIGACT-SIGMOD Symp. on Princ. Database Systems*, Waterloo, Ont., Canada, 1984, pp. 86–96.

[24] O. Ore. *Theory of graphs*. American Mathematical Society, 1962.

[25] J. Paredaens and D. Van Gucht. Possibilities and limitations of using flat operators in nested algebra expressions. In *Proc. 7th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. Database Systems*, Austin, Texas, 1988, pp. 29–38.

[26] H.-J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11, 1986, pp. 137–147.

[27] M. Stonebraker, editor. *Readings in database systems*. Morgan Kaufmann Publ., 1988.

[28] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6, 1941, pp. 73–89.

[29] A. Tarski and S. Givant. *A formalization of set theory without variables*. American Mathematical Society, Providence, Rhode Island, 1986.

[30] S.. Thomas and P.C. Fischer. Nested relational structures. In *The Theory of Databases*, P.C. Kanellakis, Ed., JAI Press, Greenwich, Conn., 1986, pp. 269–307.

[31] S.B. Zdonik and D. Maier, editors. *Readings in object-oriented database systems*. Morgan Kaufmann Publ., 1989.