Simulation of the nested relational algebra by the flat relational algebra, with an application to the complexity of evaluating powerset algebra expressions

Jan Van den Bussche
Limburgs Universitair Centrum*

April 28, 1999

Abstract

Paredaens and Van Gucht proved that the flat relational algebra has the same expressive power as the nested relational algebra, as far as queries over flat relations and with flat results are concerned. We provide a new, very direct proof of this fact using a simulation technique. Our technique is also applied to partially answer a question posed by Suciu and Paredaens regarding the complexity of evaluating powerset algebra expressions. Specifically, we show that when only unary flat relations are into play, any powerset algebra expression is either equivalent to a nested algebra expression, or its evaluation will produce intermediate results of exponential size.

Keywords: Nested relational databases, query languages, expressive power

1 Introduction

The formal basis for relational database systems is provided by the relational data model [5, 4]. A database is modeled as a collection of relations among

*Address: LUC, Departement WNI, Universitaire Campus, B-3590 Diepenbeek, Belgium. Email: vdbuss@luc.ac.be. WWW: www.luc.ac.be/~vdbuss/.

basic data values. These relations can be manipulated using five basic operators which together form the relational algebra. The *nested* relational model, designed in order to be able to represent complex data structures in a more natural and direct way [10, 3], is a typed higher-order extension of the classical "flat" relational model. In a nested relation, a tuple may consist not only of basic values but also of relations in turn. By canonically generalizing the operators of the relational algebra to work on nested relations, and adding the two operators of *nesting* and *unnesting* [9], one obtains the *nested relational algebra* [15]. These days, nested relations are known as *complex objects* [4].

The expressive power of the nested relational algebra as a query language is well understood, as well as its extensions with iteration, recursion, or the powerset operator, and extensions in the context of more general complex object data models involving not only sets but also bags, lists, arrays, and the like [4]. Two particular results we will be interested in in the present paper are those by Paredaens and Van Gucht [12], and by Suciu and Paredaens [14].

Paredaens and Van Gucht proved the *Flat-Flat Theorem*: the flat relational algebra has the same expressive power as the nested relational algebra, as far as queries over flat relations and with a flat result are concerned. Their proof was rather circuitous however, and they posed the problem of finding a direct proof. In this paper we will provide such a proof, based on a very direct simulation of the nested algebra by the flat algebra. Under this simulation, a nested relation is represented by a number of flat relations, the number depending on the scheme of the nested relation. Related simulations have been known in several variants since the '80s [2, 13, 7]. Moreover, several researchers in the field ([17, 1], see also [4, Theorem 20.7.2]) have suggested the possibility of a proof along the lines we will present. Consequently, we have written this paper not to lay any claim, but because we believe it is worthwhile to make the complete argument generally known (and actually, the detailed write-up turned out to be a rather intricate task).

Another goal of the present paper, however, is to demonstrate that the simulation technique by which we prove the flat-flat theorem can also find other applications. Specifically, we partially answer a question raised by Suciu and Paredaens concerning the complexity of evaluating powerset algebra expressions. (The powerset algebra is the extension of the nested algebra with the powerset operator.) Suciu and Paredaens conjectured that for any expression in the powerset algebra that is not equivalent to a nested algebra

expression, its evaluation will produce intermediate results of exponential size. They confirmed their conjecture for expressions defining the transitive closure of a binary relation, and more generally, for expressions defining queries on single binary relations having the form of a chain. We will confirm the conjecture for the case of multiple unary relations. The general case remains open; actually, we would not be surprised if it turned out to be false, because it is not inconceivable that there are classes of structures that are recognizable without using the powerset operator, and that have very weird combinatorial properties, such as having identifiable subparts of logarithmic size, to which we could apply the powerset operator without "blowing up," and use the result to express a query that is not expressible without the powerset.

To conclude this introduction we should mention that an analogue of the flat-flat theorem in a complex object formalism different from, but equivalent to, the nested relational model was proved by Wong using a remarkably elegant argument [18].

2 Preliminaries on nested relations

Basically we assume the existence of a countably infinite supply of *atomic attributes*. The set of atomic attributes is denoted by U. The set $\mathrm{HF}(U)$ of *hereditarily finite sets with atoms in* U is the smallest set containing U, such that if $X_1, \ldots, X_n \in \mathrm{HF}(U)$ then also $\{X_1, \ldots, X_n\} \in \mathrm{HF}(U)$. An element Ω of $\mathrm{HF}(U) - U$ is called a *scheme* if no atomic attribute occurs more than once in it.[1] Schemes are also called *complex attributes*.

Assume further given a domain V of *data values*. Let Ω be a scheme. A *relation over* Ω is a finite set of tuples over Ω. Here, a *tuple over* Ω is a mapping t on Ω, such that for $A \in \Omega \cap U$, $t(A) \in V$, and for $X \in \Omega - U$, $t(X)$ is a relation over X. We will also refer to $t(X)$ as a *complex value of type* X.

A *database scheme* is a finite set $\mathcal{S}$ of schemes. A *database over* $\mathcal{S}$ is a mapping on $\mathcal{S}$ that assigns to each scheme $\Omega \in \mathcal{S}$ a relation over Ω.

Fix a database scheme $\mathcal{S}$. The set $\mathcal{NA}$ of *nested relational algebra expressions over* $\mathcal{S}$ is inductively defined as follows. For each expression e we also define its *result scheme*, denoted by Ω_e.

[1] We say that X occurs in Ω if $X \in \Omega$, or X occurs in some $Y \in \Omega$.

- Each scheme is in $\mathcal{NA}$; its result scheme equals itself.
- If e_1 and e_2 are in $\mathcal{NA}$, with $\Omega_{e_1} = \Omega_{e_2} = \Omega$, then $(e_1 \cup e_2)$ and $(e_1 - e_2)$ are in $\mathcal{NA}$, also with result scheme Ω.
- If e_1 and e_2 are in $\mathcal{NA}$, such that no atomic attribute occurs both in Ω_{e_1} and Ω_{e_2}, then $(e_1 \times e_2)$ is in $\mathcal{NA}$, with result scheme $\Omega_{e_1} \cup \Omega_{e_2}$.
- Let e be in $\mathcal{NA}$.
  - *Projection:* if $Z \subseteq \Omega_e$, then $\pi_Z(e)$ is in $\mathcal{NA}$, with result scheme Z.
  - *Selection:* if $X, Y \in \Omega_e$ and φ is a permutation of U such that $\varphi(X) = Y$, then $\sigma_{X=_\varphi Y}(e)$ is in $\mathcal{NA}$, with result scheme Ω_e.
  - *Renaming:* if φ is a permutation of U, then $\rho_\varphi(e)$ is in $\mathcal{NA}$, with result scheme $\varphi(\Omega_e)$.
  - *Nesting:* if $Z \subseteq \Omega_e$, then $\nu_Z(e)$ is in $\mathcal{NA}$, with scheme $(\Omega_e - Z) \cup \{Z\}$.
  - *Unnesting:* if $X \in \Omega_e - U$, then $\mu_X(e)$ is in $\mathcal{NA}$, with scheme $(\Omega_e - \{X\}) \cup X$.

Let Δ be a database over $\mathcal{S}$, and let e be a nested relational algebra expression over $\mathcal{S}$. The *result of evaluating* e *on* Δ, denoted by $e(\Delta)$, is inductively defined as follows:

- If e is a scheme Ω, then $e(\Delta) := \Delta(\Omega)$.
- $(e_1 \cup e_2)(\Delta) := e_1(\Delta) \cup e_2(\Delta)$; $(e_1 - e_2)(\Delta) := e_1(\Delta) - e_2(\Delta)$.
- $(e_1 \times e_2)(\Delta) := \{t_1 \cup t_2 \mid t_1 \in e_1(\Delta) \text{ and } t_1 \in e_2(\Delta)\}$.
- $\pi_Z(e)(\Delta) := \{t|_Z \mid t \in e(\Delta)\}$.
- $\sigma_{X=_\varphi Y}(e)(\Delta) := \{t \in e(\Delta) \mid \varphi(t(X)) = t(Y)\}$.[2]
- $\rho_\varphi(e)(\Delta) := \{\varphi(t) \mid t \in e(\Delta)\}$.

[2]Permutations of U are applied to tuples and relations in the obvious way. If t is a tuple over Ω, then $\varphi(t)$ is the tuple over $\varphi(\Omega)$ defined by $\varphi(t)(\varphi(X)) = t(X)$.

- $\nu_Z(e)(\Delta) := \{t$ tuple over $(\Omega - Z) \cup \{Z\} \mid \exists t' \in e(\Delta) : t|_{\Omega-Z} = t'|_{\Omega-Z}$ and $t(Z) = \{t''|_Z \mid t'' \in e(\Delta)$ and $t''|_{\Omega-Z} = t'|_{\Omega-Z}\}\}$.
- $\mu_X(e)(\Delta) := \{t$ tuple over $(\Omega - \{X\}) \cup X \mid \exists t' \in e(\Delta) : t|_{\Omega-\{X\}} = t'|_{\Omega-\{X\}}$ and $t|_X \in t'(X)\}$.

So, a nested relational algebra expression with result scheme Ω defines a mapping from databases to relations over Ω. Such mappings are called *queries*.

The φ in a selection operation $\sigma_{X=_\varphi Y}$ is important when X and Y are complex, because it specifies how the two complex values $t(X)$ and $t(Y)$, for some tuple t, are to be compared. For example, if $X = \{A, B\}$ and $Y = \{C, D\}$, then $\sigma_{X=_{\varphi_1} Y}$, where $\varphi_1(A) = C$ and $\varphi_1(B) = D$, has a different semantics than $\sigma_{X=_{\varphi_2} Y}$, where $\varphi_2(A) = D$ and $\varphi_2(B) = C$. When X and Y are atomic, the φ is irrelevant (there is only one way to compare two atomic values) and we will omit it.

3 Representing nested relations by flat databases

A scheme is called *flat* if all its elements are atomic attributes. A database scheme is called flat if all its schemes are flat.

In order to formally define a representation of nested relations by flat databases, we must first make some technical assumptions about the set U of atomic attributes. We partition the atomic attributes in "ordinary" attributes and "identifier" attributes. Unless explicitly specified otherwise, an atomic attribute is always assumed to be ordinary, and a scheme is always assumed to be built up from ordinary atomic attributes only. For each scheme X, we assume we have the infinitely many identifier attributes

$$\mathrm{id}(X)_1,\ \mathrm{id}(X)_2,\ \mathrm{id}(X)_3,\ \ldots$$

such that if $X \neq Y$ or $i \neq j$ then $\mathrm{id}(X)_i \neq \mathrm{id}(Y)_j$.

Our representation of a nested relation by a flat databases uses identifiers for complex values. These identifiers are tuples of atomic values occurring in the nested relation. The width of these tuples can vary depending on the type of complex value represented. Apart from the flat relation representing the nested relation itself, the flat database will have auxiliary relations, one for each complex attribute X, holding the identifiers representing a complex

value of type X, and specifying which complex values are represented by these identifiers.

Why do we need identifiers that are tuples? Why can't we just use identifiers that are atomic values? The point is that later we want to simulate the nested relational algebra by the flat relational algebra. Operations that introduce new complex values, notably nesting, will have to be simulated by introducing identifiers for these new complex values. The relational algebra cannot "invent" new atomic values. Hence, it has to construct the identifiers as tuples of the atomic values existing in the database. If we could only use identifiers of length 1, and there are n distinct atomic values in the database, we would only have n different identifiers at our disposition, which is much too little. For example, the following nested relational algebra expression, starting from a flat relation over $\{A\}$, introduces n^2 new complex values of type $\{C, D\}$: one for each pair of atomic values.[3]

$$\nu_{\{A,B\}}\sigma_{B=D}\sigma_{A=C}(\{A\} \times \rho_{(A\ B)}(\{A\}) \times \rho_{(A\ C)}(\{A\}) \times \rho_{(A\ D)}(\{A\}))$$

The lengths of the identifiers, depending on the type of complex value they represent, are given by an identifier-width assignment, defined next:

Definition 1 Let Ω be a scheme. An *identifier-width assignment (i.w.a.) over* Ω is a mapping α from the set of complex attributes occurring in Ω to the natural numbers.

For a complex attribute X occurring in Ω, we will denote the set

$$\{\mathrm{id}(X)_1, \ldots, \mathrm{id}(X)_{\alpha(X)}\}$$

by $\mathrm{ID}_\alpha(X)$. This set contains the atomic attributes of the identifier tuples for complex values of type X.

Given a scheme Ω and an i.w.a. α over Ω, we can now define the flat database scheme listing the flat schemes of the flat relations that together make up a flat database representation of a nested relation over Ω. This flat database scheme is denoted by $\mathrm{flat}_\alpha(\Omega)$.

Definition 2 The flat database scheme $\mathrm{flat}_\alpha(\Omega)$ consists the scheme

$$\mathrm{rep}_\alpha(\Omega) := (\Omega \cap U) \cup \bigcup\{\mathrm{ID}_\alpha(X) \mid X \in \Omega - U\},$$

[3]We use the standard notation $(A\ B)$ for the permutation that exchanges A and B.

together with, for all complex attributes X occurring in Ω, the schemes

$$\mathrm{rep}_\alpha(X) := (X \cap U) \cup \bigcup \{\mathrm{ID}_\alpha(Y) \mid Y \in X - U\} \cup \mathrm{ID}_\alpha(X).$$

Flat databases over $\mathrm{flat}_\alpha(\Omega)$ represent nested relations over Ω. To define this representation formally in Definition 4, we need the following auxiliary technical definition:

Definition 3 Let Ω be a scheme, let X be a complex attribute occurring in Ω, and let α be an i.w.a. over Ω. Let Δ be a database over $\mathrm{flat}_\alpha(\Omega)$ and let t be a tuple over $\mathrm{ID}_\alpha(X)$. Let α_X be the restriction of α to the complex attributes occurring in X. Then Δ_t is the database over $\mathrm{flat}_{\alpha_X}(X)$ defined by

$$\Delta_t(\mathrm{rep}_{\alpha_X}(X)) := \{t'|_{\mathrm{rep}_{\alpha_X}(X)} \mid t' \in \Delta(\mathrm{rep}_\alpha(X)) \text{ and } t'|_{\mathrm{ID}_\alpha(X)} = t\},$$

and, for each complex attribute Y occurring in X,

$$\Delta_t(\mathrm{rep}_{\alpha_X}(Y)) := \Delta(\mathrm{rep}_\alpha(Y)).$$

To clarify the notation used in the above definition, it should be pointed out that $\mathrm{rep}_{\alpha_X}(X)$ is *not* the same as $\mathrm{rep}_\alpha(X)$; the latter includes $\mathrm{ID}_\alpha(X)$ as a subset, while the former doesn't, because for α_X, X is the top-level scheme. We see clearly in Definition 2 that $\mathrm{rep}_\alpha(Z)$ only contains $\mathrm{ID}_\alpha(Z)$ as a subset if Z is a lower-level scheme (a complex attribute occurring in the top-level scheme).

Of course, for the Ys in the above definition, which are lower-level even when viewing X as the top-level, $\mathrm{rep}_{\alpha_X}(Y)$ *is* the same as $\mathrm{rep}_\alpha(Y)$, which is why at these Y the definition of Δ_t is simpler.

We can now define:

Definition 4 Let Ω be a scheme, let α be an i.w.a. over Ω, and let Δ be a database over $\mathrm{flat}_\alpha(\Omega)$. The *nested relation represented by* Δ, denoted by $\mathrm{nested}(\Delta)$, equals

$$\{t \text{ tuple over } \Omega \mid \exists t' \in \Delta(\mathrm{rep}_\alpha(\Omega)) : t|_{\Omega \cap U} = t'|_{\Omega \cap U} \text{ and } \forall X \in \Omega - U : t(X) = \mathrm{nested}(\Delta_{t'|_{\mathrm{ID}_\alpha(X)}})\}.$$

Note how the tuple $t'|_{\mathrm{ID}_\alpha(X)}$ serves as an identifier for the complex value $\mathrm{nested}(\Delta_{t'|_{\mathrm{ID}_\alpha(X)}})$.

As a (perhaps too trivial) example, let $\Omega = \{\{A\}\}$ with $A \in U$, and let $\alpha(\{A\}) = 1$. Then $\mathrm{flat}_\alpha(\Omega)$ consists of $\mathrm{rep}_\alpha(\Omega) = \{\mathrm{id}(\{A\})_1\}$ and $\mathrm{rep}_\alpha(\{A\}) = \{\mathrm{id}(\{A\})_1, A\}$. Let Δ be the following database over $\mathrm{flat}_\alpha(\Omega)$:

{ id($\{A\}$)$_1$ }
a
b
c
d

{ id($\{A\}$)$_1$	A }
a	a
a	b
a	c
b	c
b	d
c	b
d	b

Then $\mathrm{nested}(\Delta)$ equals

{ $\{A\}$ }
a
b
c
c
d
b

As a final remark we note that occurrences of the empty complex value of some type X would also be represented by an identifier, but this identifier would not show up in the corresponding $\mathrm{rep}_\alpha(X)$ relation since it represents the empty set.

4 Simulation of nested algebra by flat algebra

A nested relational algebra expression is called *flat* if it is defined over a flat database scheme and does not use the nesting (ν) and the unnesting (μ) operators.

In this section we will prove:

Theorem 1 *Let e be a nested relational algebra expression over a flat database scheme $\mathcal{S}$. Then there exists an i.w.a. α over Ω_e and flat relational*

algebra expressions e_X *over* $\mathcal{S}$ *for* $X = \Omega_e$ *or* X *a complex attribute occurring in* Ω_e*, such that for each database* Δ *over* $\mathcal{S}$*,*

$$\text{nested}(\Delta_e) = e(\Delta),$$

where Δ_e *is the database over* $\text{flat}_\alpha(\Omega_e)$ *defined by* $\Delta_e(\text{rep}_\alpha(X)) = e_X(\Delta)$ *for each* X*.*

As a corollary, we get:

Theorem 2 (Paredaens-Van Gucht) *Let* e *be a nested relational algebra expression over a flat database scheme* $\mathcal{S}$*, such that* Ω_e *is flat. Then there exists a flat relational algebra expression* e' *over* $\mathcal{S}$ *such that for each database* Δ *over* $\mathcal{S}$*,*

$$e'(\Delta) = e(\Delta).$$

Proof. Since Ω_e is flat, $\text{flat}_\alpha(\Omega_e)$ consists simply of Ω_e itself and for any database Δ' over $\text{flat}_\alpha(\Omega_e)$, $\text{nested}(\Delta') = \Delta'(\Omega_e)$. Theorem 1 thus tells us there exists a flat relational algebra expresion $e' = e_{\Omega_e}$ such that for each Δ, $e'(\Delta) = e(\Delta)$, as desired. ■

Before we prove Theorem 1, we illustrate the most intricate part of the proof, the simulation of nesting, with a simple example.

Let $\mathcal{S}$ consist of the single relation scheme $\Omega = \{A, B\}$. Consider the database Δ over $\mathcal{S}$ defined by

$$\Delta(\Omega) = \begin{array}{|cc|}\hline \{\ A & B\ \} \\ \hline a & b \\ b & b \\ a & c \\ b & c \\ c & d \\ \hline \end{array}.$$

First, consider the expression $\nu_{\{A\}}(\Omega)$. Evaluating it on Δ yields the following relation:

$$\begin{array}{|cc|}\hline \{\ \{A\} & B\ \} \\ \hline \begin{array}{|c|}\hline a \\ b \\ \hline \end{array} & b \\ \begin{array}{|c|}\hline a \\ b \\ \hline \end{array} & c \\ \begin{array}{|c|}\hline c \\ \hline \end{array} & d \\ \hline \end{array}$$

We can represent this relation by the following flat database:

$\{\ \mathrm{id}(\{A\})_1$	$B\ \}$
b	b
c	c
d	d

$\{\ \mathrm{id}(\{A\})_1$	$A\ \}$
b	a
b	b
c	a
c	b
d	c

Next, consider the expression $\nu_{\{B\}}\nu_{\{A\}}(\Omega)$. Its evaluation on Δ yields the following relation:

$\{\ \{A\}$	$\{B\}\ \}$
a b	b c
c	d

This relation can be represented by the following flat database:

$\{\ \mathrm{id}(\{A\})_1$	$\mathrm{id}(\{B\})_1\ \}$
b	b
c	c
d	d

$\{\ \mathrm{id}(\{A\})_1$	$A\ \}$
b	a
b	b
c	a
c	b
d	c

$\{\ \mathrm{id}(\{B\})_1$	$B\ \}$
b	b
b	c
c	b
c	c
d	d

■

We now present:

Proof of Theorem 1. In the proof, we will rely on the well-known fact that the flat relational algebra is as powerful as the *tuple relational calculus* [16, 11], a variant of first-order logic.[4] So instead of writing algebra expressions we will often write calculus formulas whenever this is more convenient.

We begin by stating the following:

Lemma 1 *Let Ω be a scheme, let X be a complex attribute occurring in Ω, and let α be an i.w.a. over Ω. Then there exists a tuple relational calculus formula $equal_\alpha(t_1, t_2)$ such that for each database Δ over $\mathrm{flat}_\alpha(\Omega)$ and tuples t_1, t_2 over $\mathrm{ID}_\alpha(X)$, $equal_\alpha(t_1, t_2)$ is true in Δ iff $\mathrm{nested}(\Delta_{t_1}) = \mathrm{nested}(\Delta_{t_2})$.*

[4] This is with the understanding that we use the active-domain semantics of the calculus [4].

In the formulation of the above lemma, note that Δ_{t_1} and Δ_{t_2} are flat databases over $\mathrm{flat}_{\alpha_X}(X)$, and thus $\mathrm{nested}(\Delta_{t_1})$ and $\mathrm{nested}(\Delta_{t_2})$ are complex values of type X.

Proof of Lemma 1. We inductively construct a formula

$$subset_\alpha(t_1, t_2)$$

and define $equal_\alpha(t_1, t_2)$ as $subset_\alpha(t_1, t_2) \wedge subset_\alpha(t_2, t_1)$.

The formula $subset_\alpha(t_1, t_2)$ is defined as:

$$\begin{aligned}\forall t \in \mathrm{rep}_\alpha(X) : t|_{\mathrm{ID}_\alpha(X)} = t_1 \Rightarrow \\ \exists t' \in \mathrm{rep}_\alpha(X) : t'|_{\mathrm{ID}_\alpha(X)} = t_2 \\ \wedge\, t'|_{X\cap U} = t|_{X\cap U} \wedge \bigwedge_{Y\in X-U} equal_\alpha(t'|_{\mathrm{ID}_\alpha(Y)}, t|_{\mathrm{ID}_\alpha(Y)}).\end{aligned}$$ ■

For any scheme Z, we will use the abbreviation $equiv_\alpha(Z)(t_1, t_2)$ for the formula

$$t_1|_{Z\cap U} = t_2|_{Z\cap U} \wedge \bigcup_{X\in Z-U} equal_\alpha(t_1|_{\mathrm{ID}_\alpha(X)}, t_2|_{\mathrm{ID}_\alpha(X)}).$$

This formula clearly expresses that the flat tuples t_1 and t_2 represent the same nested tuple of type Z.

The actual proof of Theorem 1 now proceeds by induction on the structure of the expression e.

- e is a (flat) scheme Ω. In this basic case we define

  $$e_\Omega := \Omega.$$

  (There is no need to define α as there are no complex attributes occurring in Ω.)

- $e = \nu_Z(e')$. By induction, we have an i.w.a. α' over $\Omega_{e'}$ satisfying the theorem. Let us denote

  $$W := ((\Omega_{e'} - Z) \cap U) \cup \bigcup\{\mathrm{ID}_{\alpha'}(X) \mid X \in (\Omega_{e'} - Z) - U\}.$$

  Define $\alpha(Z) := |W|$ and $\alpha(Y) := \alpha'(Y)$ for all other complex attributes occurring in Ω_e. Let φ be a bijection from W to $\mathrm{ID}_\alpha(Z)$. Then we define

  $$e_{\Omega_e} := \{t \cup \varphi(t) \mid t \in \pi_W(e'_{\Omega_{e'}})\},$$

and

$$e_Z := \{t \text{ tuple over } \mathrm{rep}_\alpha(Z) \mid \exists t', t'' \in \pi_W(e'_{\Omega_{e'}}) : t' = \varphi^{-1}(t|_{\mathrm{ID}_\alpha(Z)}) \wedge equiv_{\alpha'}(\Omega_{e'} - Z)(t', t'') \wedge (t|_{\mathrm{rep}_\alpha(Z) - \mathrm{ID}_\alpha(Z)} \cup t'') \in e'_{\Omega_{e'}}\},$$

with the understanding that in the formula $equiv_{\alpha'}(\Omega_{e'} - Z)$, every occurrence of a scheme $\mathrm{rep}_{\alpha'}(Y)$ (for some Y) is replaced by the corresponding expression e'_Y. Finally, for any other complex attribute $X \neq Z$ occurring in e_{Ω_e}, define $e_X := e'_X$.

- $e = \mu_X(e')$. Define α as the restriction of α' to the complex attributes occurring in Ω_e. Let φ be a permutation of U such that $\varphi(\mathrm{ID}_{\alpha'}(X))$ is disjoint from $\mathrm{rep}_{\alpha'}(\Omega_{e'}) \cup \mathrm{rep}_{\alpha'}(X)$. Define

$$e_{\Omega_e} := \pi_{\mathrm{rep}_\alpha(\Omega_e)} \sigma_{\mathrm{id}(X)_1 = \varphi(\mathrm{id}(X)_1)} \ldots \sigma_{\mathrm{id}(X)_{\alpha'(X)} = \varphi(\mathrm{id}(X)_{\alpha'(X)})} (e'_{\Omega_{e'}} \times \rho_\varphi(e'_X))$$

  and define $e_Y := e'_Y$ for complex attributes Y occurring in Ω_e.

- $e = (e_1 \times e_2)$. By induction we have i.w.a.'s α_1 for e_1 and α_2 for e_2 satisfying the theorem. For a complex attribute X occurring in Ω_e, define $\alpha(X) := \alpha_1(X)$ and $e_X := (e_1)_X$ if X occurs in Ω_{e_1}, and $\alpha(X) := \alpha_2(X)$ and $e_X := (e_2)_X$ if X occurs in Ω_{e_2} (by the syntax of the nested relational algebra exactly one of the two cases holds). Define $e_{\Omega_e} := (e_1)_{\Omega_{e_1}} \times (e_2)_{\Omega_{e_2}}$.

- $e = (e_1 \cup e_2)$. Define

$$\alpha(X) := \max\{\alpha_1(X), \alpha_2(X)\} + 2$$

  and

$$e_X := (e_1)'_X \cup (e_2)'_X,$$

  where $(e_1)'_X$ and $(e_2)'_X$ are defined as follows.

  $(e_1)'_X$ is obtained by taking the Cartesian product ($\times$) of $(e_1)_X$ with the following factors for all $Y \in (X - U)$, as well as, if $X \neq \Omega_e$, for

$Y = X$ itself:

$$\{t \text{ tuple over } \{\mathrm{id}(Y)_{\alpha_1(Y)+1}, \mathrm{id}(Y)_{\alpha_1(Y)+2}, \ldots, \mathrm{id}(Y)_{\alpha(Y)}\} \mid \\ t(\mathrm{id}(Y)_{\alpha_1(Y)+1}) \in \mathrm{adom} \\ \wedge\, t(\mathrm{id}(Y)_{\alpha_1(Y)+1}) = t(\mathrm{id}(Y)_{\alpha_1(Y)+2}) = \cdots = t(\mathrm{id}(Y)_{\alpha(Y)})\}.$$

$(e_2)'_X$ is obtained by taking the Cartesian product ($\times$) of $(e_2)_X$ with the following factors for all $Y \in (X - U)$, as well as, if $X \neq \Omega_e$, for $Y = X$ itself:

$$\{t \text{ tuple over } \{\mathrm{id}(Y)_{\alpha_2(Y)+1}, \mathrm{id}(Y)_{\alpha_2(Y)+2}, \ldots, \mathrm{id}(Y)_{\alpha(Y)}\} \mid \\ t(\mathrm{id}(Y)_{\alpha_2(Y)+1}) \neq t(\mathrm{id}(Y)_{\alpha_2(Y)+2}) \in \mathrm{adom} \\ \wedge\, t(\mathrm{id}(Y)_{\alpha_2(Y)+2}) = t(\mathrm{id}(Y)_{\alpha_2(Y)+3}) = \cdots = t(\mathrm{id}(Y)_{\alpha(Y)})\}.$$

These Cartesian products make sure that identifier tuples appearing in the result of evaluating $(e_1)'_X$ will be different from identifier tuples appearing in the result of evaluating $(e_2)'_X$, so that no mix-up occurs when taking the union.[5]

- $e = (e_1 - e_2)$. This case is exactly the same as the case $e = (e_1 \cup e_2)$, except that we define e_{Ω_e} as

$$\{t \in (e_1)'_{\Omega_e} \mid \neg \exists t' \in (e_2)'_{\Omega_e} : \mathit{equiv}_\alpha(\Omega_e)(t, t')\},$$

with the understanding that in the formula $\mathit{equiv}_\alpha(\Omega_e)$, every occurrence of a scheme $\mathit{rep}_\alpha(Y)$ (for some Y) is replaced by the corresponding expression e_Y.

- $e = \pi_Z(e')$. Then α equals the restriction of α' to the complex attributes occurring in Z,

$$e_{\Omega_e} := \pi_{\mathrm{rep}_\alpha(Z)}(e'_{\Omega_{e'}}),$$

and $e_X := e'_X$ for each complex attribute X occurring in Z.

[5]This trick does not work if adom would contain only one element. But this is harmless, because we can treat this case (as well as the case where adom is completely empty) entirely separately, because in this case there are, up to isomorphism, only a finite number of possible databases: we can test for these possibilities and return, for each possibility, directly the right result, bypassing the simulation.

- $e = \sigma_{X=_\varphi Y}(e')$. Take $\alpha := \alpha'$ and define $e_Z := e'_Z$ for each complex attribute Z occurring in Ω_e. If X and Y are atomic, e_{Ω_e} is simply $\sigma_{X=Y}(e'_{\Omega_{e'}})$. If X and Y are complex, we need the following lemma similar to Lemma 1: (the proof is analogous)

Lemma 2 *Let Ω be a scheme, let $X, Y \in \Omega - U$, let φ be a permutation of U such that $\varphi(X) = Y$, and let α be an i.w.a. over Ω. Then there exists a tuple relational calculus formula $equal_\alpha(t_1, t_2)$ such that for each database Δ over $\mathrm{flat}_\alpha(\Omega)$, tuple t_1 over $\mathrm{ID}_\alpha(X)$, and tuple t_2 over $\mathrm{ID}_\alpha(Y)$, $equal_\alpha(t_1, t_2)$ is true in Δ iff $\varphi(\mathrm{nested}(\Delta_{t_1})) = \mathrm{nested}(\Delta_{t_2})$.*

We then define

$$e_{\Omega_e} := \{t \in e'_{\Omega'_e} \mid equal_{\alpha'}(t|_{\mathrm{ID}_{\alpha'}(X)}, t|_{\mathrm{ID}_{\alpha'}(Y)})\}.$$

- $e = \rho_\varphi(e')$. Extend φ to identifier attributes in the following canonical manner: $\varphi(\mathrm{id}(X)_i) := \mathrm{id}(\varphi(X))_i$. Then $\alpha(\varphi(X)) := \alpha'(X)$ and $e_X := \rho_\varphi(e'_X)$ for $X = \Omega_e$ or X a complex attribute occurring in Ω_e. ■

5 Complexity of evaluating powerset algebra expressions over unary relations

The *powerset algebra* is the extension of the nested relational algebra with the powerset operator (Π). Syntactically, if e is an expression, then $\Pi(e)$ is also an expression, with output scheme $\{\Omega_e\}$. Semantically, on any database Δ, $\Pi(e)(\Delta)$ equals $\{t$ tuple over $\{\Omega_e\} \mid t(\Omega_e) \subseteq e(\Delta)\}$.

Hull and Su [8] showed that in the powerset algebra precisely all queries computable in elementary time are expressible. So the powerset algebra is a very powerful query language. It does not seem to be a very practical language, however, in the sense that no example is known of a query not expressible in the nested relational algebra, that is expressible in the powerset algebra by an expression whose evaluation never generates intermediate results of exponential size (in spite of applications of the powerset operator).

In this section, we prove that no such query exists if it is over a *unary* database scheme. A database scheme is called unary if all its schemes are singletons of the form $\{A\}$, with A atomic.

The proof will be easy once we have established the following:

Lemma 3 *Let $\mathcal{S}$ be a unary database scheme and let e be a nested relational algebra expression over $\mathcal{S}$. Let $|e| : \mathbf{N} \to \mathbf{N}$ be the mapping on the natural numbers defined as follows: $|e|(n)$ is the maximal cardinality of $e(\Delta)$, where Δ is a database over $\mathcal{S}$ with active domain of cardinality n. Then either $|e| = O(1)$ or $|e| = \Omega(n)$.*

The active domain of a database Δ, denoted by $\mathrm{adom}(\Delta)$, is the set of all data values appearing in the relations of the database.

Proof of Lemma 3. Apply Theorem 1 to obtain an i.w.a. α over Ω_e and flat relational algebra expressions e_X which simulate e in the sense described by the theorem.

Let Δ be a database over $\mathcal{S}$. Let Δ_e be the database over $\mathrm{flat}_\alpha(\Omega_e)$ described by Theorem 1. Consider the following equivalence relation $\equiv_\Delta$ on tuples in $\Delta_e(\mathrm{rep}_\alpha(\Omega_e))$: $t_1 \equiv_\Delta t_2$ if $equiv_\alpha(\Omega_e)(t_1, t_2)$ holds in Δ_e.[6] So, $t_1 \equiv_\Delta t_2$ iff t_1 and t_2 represent the same nested tuple in $e(\Delta)$. Hence, the cardinality of $e(\Delta)$ equals $|\equiv_\Delta|$, the index (number of different equivalence classes) of $\equiv_\Delta$.[7]

Since the relations of Δ_e can be computed by the same expressions e_X for any given Δ, and since $equiv_\alpha(\Omega_e)$ is a tuple relational calculus formula, there is one tuple relational calculus formula $\varphi(t_1, t_2)$ such that for every database Δ, $t_1 \equiv_\Delta t_2$ iff $\varphi(t_1, t_2)$ holds in Δ.

Since the tuple relational calculus is equivalent to the domain relational calculus (essentially first-order logic) [16], we can equivalently express φ as a *domain*, rather than a *tuple*, relational calculus formula (which we also denoted by φ by abuse of notation). Variables now range over the active domain of the input database.

A database Δ over $\mathcal{S}$ consists of a set of relations over unary schemes. We naturally view a relation over a unary scheme $\{A\}$ as a set of data values (formally it is a set of mappings from $\{A\}$ to V). The unary relations of Δ induce a partition on $\mathrm{adom}(\Delta)$. Each partition class is determined by some non-empty subset $X \subseteq \mathcal{S}$ and equals

$$\Delta[X] := \bigcap_{Y \in X} \Delta(Y) - \bigcup_{Y \in \mathcal{S} - X} \Delta(Y).$$

The automorphisms of Δ are precisely the permutations of $\mathrm{adom}(\Delta)$ that leave every $\Delta[X]$ invariant.

[6] The definition of $equiv_\alpha(Z)$ was given after Lemma 1.

[7] We denote the index of an equivalence relation R by $|R|$.

Let k be the number of different variables used in the formula φ. Let us call any function δ from the non-empty subsets of $\mathcal{S}$ to $\{0, \ldots, k\}$ a *classifier*. We classify the databases over $\mathcal{S}$ using these classifiers as follows: for a classifier δ, $\mathcal{S}_\delta$ denotes the family of all databases Δ over $\mathcal{S}$ for which

$$\text{cardinality of } \Delta[X] \begin{cases} = \delta(X) & \text{if } \delta(X) < k, \text{ and} \\ \geq k & \text{if } \delta(X) = k \end{cases} \qquad \text{for each } X.$$

The families $\mathcal{S}_\delta$ with a δ such that $\delta(X) < k$ for every X are finite (up to isomorphism) and can be discarded. If we can show for all other δ that $|\equiv|$ restricted to $\mathcal{S}_\delta$ is either $O(1)$ or $\Omega(n)$, we are ready. Indeed, if all of them are $O(1)$, then also globally $|\equiv|$ is $O(1)$; if at least one of them is $\Omega(n)$, then also globally $|\equiv|$ is $\Omega(n)$ since the families are infinite (we just discarded the finite ones).

So fix a family $\mathcal{S}_\delta$; we only consider databases in this family. It is a routine exercise in logic (compare Exercise 1.3.11 in [6]) to show that any formula over $\mathcal{S}$ that uses at most k different variables is equivalent, on $\mathcal{S}_\delta$, to a quantifier-free formula. This holds in particular for formula φ. We may assume without loss of generality that φ is in disjunctive normal form, and that each conjunction in this disjunction is maximally consistent. Note that a maximally consistent conjunction of literals over $\mathcal{S}$, with m variables, serves as an *automorphism type* (also called *m-type*): it describes an m-tuple of data values entirely up to application of an automorphism, specifying the partition class of every component, as well as all equalities and non-equalites that hold among the components. In the case of φ, m equals 2ℓ, where ℓ is the cardinality of $\mathrm{rep}_\alpha(\Omega_e)$. Note also that a 2ℓ-type is nothing but the conjunction of two ℓ-types and the specification of the equalities and non-equalities holding across these two types.

We distinguish between the following possibilities:

- φ is the empty disjunction, i.e., equivalent to false. Then $\equiv_\Delta$ is the empty equivalence relation on each Δ. But since $\equiv_\Delta$ is defined on $\mathrm{rep}_{\Omega_e}(\Delta_e)$, i.e., on e_{Ω_e}, this means that $e_{\Omega_e}(\Delta)$, and thus also $e(\Delta)$, is empty on each Δ. In this case $|e|$ is everywhere zero and thus trivially $O(1)$.

- φ is not false, and there is an ℓ-type τ such that the 2ℓ-type describing those pairs of tuples (t_1, t_2) such that

  - t_1 and t_2 both satisfy τ;

– t_1 and t_2 are equal outside Z_τ; and

– t_1 and t_2 are disjoint on Z_τ,

is *not* in φ. Here, Z_τ is the set of those attributes for which the corresponding variable in τ is specified by τ to take a value in the partition class determined by an $X \subseteq \mathcal{S}$ with $\delta(X) = k$.

So for any database Δ and any pair (t_1, t_2) of tuples in Δ as above, $t_1 \not\equiv_\Delta t_2$. By augmenting Δ with a fixed number of new data values, placed in the appropriate partition classes, we get a third tuple t_3 not equivalent to t_1 or t_2. We can keep on doing this, so that $|\equiv|$ is $\Omega(n)$.

- φ is not false, and there is no such ℓ-type τ as in the previous item. But then, for large enough Δ, any two tuples t_1, t_2 of the same ℓ-type τ that are equal outside Z_τ are equivalent. Indeed, we can always find a third tuple t_3 of the same type disjoint from both t_1 and t_2 on Z_τ but equal outside; by assumption we then have $t_1 \equiv_\Delta t_3 \equiv_\Delta t_2$ and thus $t_1 \equiv_\Delta t_2$. Hence, in this case $|\equiv|$ is $O(1)$, being bounded by the number of different ℓ-types, which is a fixed number, and the number of different values a tuple of some type τ can have outside Z_τ, which is also fixed by definition of Z_τ (components outside Z_τ belong to a partition class determined by an $X \subseteq \mathcal{S}$ with $\delta(X) < k$ and thus of fixed size). ■

As a corollary, we get:

Theorem 3 *Let e be a powerset algebra expression over a unary database scheme $\mathcal{S}$. Then either e is equivalent to a nested relational algebra expression, or for some subexpression e' of e, $|e'|$ is $\Omega(2^n)$.*

Proof. Let e' be a minimal subexpression of e of the form $\Pi(e'')$. By Lemma 3, $|e''|$ is either $O(1)$ or $\Omega(n)$. If $|e''|$ is $\Omega(n)$, then clearly $|e'|$ is $\Omega(2^n)$.

If $|e''|$ is $O(1)$, we can simulate the application of the powerset operator in the nested relational algebra. Indeed, let K be the maximal cardinality of

$e''(\Delta)$. Then e' is equivalent to the following expression: (let $\Omega := \Omega_{e''}$)

$$\pi_{\{\Omega\}}\nu_{\Omega} \quad \Big(\quad \begin{array}{c} \sigma_{\vec{\Omega}=\varphi_1(\vec{\Omega})}(e'' \times \rho_{\varphi_1}(e'') \times \cdots \times \rho_{\varphi_K}(e'')) \\ \cup \\ \vdots \\ \cup \\ \sigma_{\vec{\Omega}=\varphi_K(\vec{\Omega})}(e'' \times \rho_{\varphi_1}(e'') \times \cdots \times \rho_{\varphi_K}(e'')) \end{array} \quad \Big),$$

where $\varphi_1, \ldots, \varphi_K$ are permutations of U such that every atomic attribute occurs in at most one of Ω, $\varphi_1(\Omega)$, ..., $\varphi_K(\Omega)$, and $\sigma_{\vec{\Omega}=\varphi(\vec{\Omega})}$ is an abbreviation for the sequence of all $\sigma_{X=\varphi\varphi(X)}$ with $X \in \Omega$. ■

References

[1] S. Abiteboul. Personal communication, 1993.

[2] S. Abiteboul and N. Bidoit. Non first normal form relations: An algebra allowing data restructuring. *Journal of Computer and System Sciences*, 33(3):361–393, 1986.

[3] S. Abiteboul, P.C. Fischer, and H.-J. Schek, editors. *Nested Relations and Complex Objects in Databases*, volume 361 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[5] E. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970.

[6] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.

[7] M. Gyssens, D. Suciu, and D. Van Gucht. The restricted and the bounded fixpoint closures of the nested relational algebra are equivalent. In P. Atzeni and V. Tannen, editors, *Database Programming Languages (DBPL-5)*, Electronic Workshops in Computing. Springer-Verlag, 1995. http://www.springer.co.uk/ewic/workshops/DBPL5/.

[8] R. Hull and J. Su. On the expressive power of database queries with intermediate types. *Journal of Computer and System Sciences*, 43(1):219–237, 1991.

[9] G. Jaeshke and H.-J. Schek. Remarks on the algebra of non-first normal form relations. In *Proceedings of the First ACM Symposium on Principles of Database Systems*, pages 124–138. ACM Press, 1982.

[10] A. Makinouchi. A consideration of normal form of not-necessarily-normalized relations in the relation data model. In *Proceedings 3th International Conference on Very Large Data Bases*, volume 9:4 of *SIGMOD Record*, pages 447–453, 1977.

[11] J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The Structure of the Relational Database Model*, volume 17 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1989.

[12] J. Paredaens and D. Van Gucht. Converting nested algebra expressions into flat algebra expressions. *ACM Transactions on Database Systems*, 17(1):65–93, 1992.

[13] D. Suciu. Bounded fixpoints for complex objects. *Theoretical Computer Science*, 176(1–2):283–328, 1997.

[14] D. Suciu and J. Paredaens. The complexity of the evaluation of complex algebra expressions. *Journal of Computer and System Sciences*, 55(2):322–343, 1997.

[15] S. Thomas and P. Fischer. Nested relational structures. In P. Kanellakis, editor, *The Theory of Databases*, pages 269–307. JAI Press, 1986.

[16] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.

[17] D. Van Gucht. Personal communication, 1990.

[18] L. Wong. Normal forms and conservative extension properties for query languages over collection types. *Journal of Computer and System Sciences*, 52(3):495–505, 1996.