Solving equations in the relational algebra

Joachim Biskup, U. Dortmund
Jan Paredaens, U. Antwerp
Thomas Schwentick, U. Mainz
Jan Van den Bussche, U. Limburg

Abstract

Enumerating all solutions of a relational algebra equation is a natural and powerful operation which, when added as a query language primitive to the nested relational algebra, yields a query language for nested relational databases, equivalent to the well-known powerset algebra. We study *sparse* equations, which are equations with at most polynomially many solutions. We look at their complexity, and compare their expressive power with that of similar notions in the powerset algebra.

1 Introduction

Suppose we are allowed to see only a view on a database B, computed by a relational algebra expression e. If we still want to find out what B is, we might try to "invert" e (assuming we know this expression), which will only work when we also know the finite domain D of B. Specifically, we can enumerate all databases X over D, and test for each X whether it satisfies the equation $e(X) = e(B)$. One of these solutions will be B of course, so if the set of all solutions is not too big, it might provide us with useful information to start our detective work.

The above simple scenario from database security led us to wonder what can be said in general about the solution of equations in the relational algebra. Generally, if e_1 and e_2 are two algebra expressions over the database schema augmented with some relation variables X_1, … , X_p, we can consider the equation $e_1 = e_2$. A *solution* of this equation, given a database B with finite domain D, is a tuple $(X_1, \dots, X_p)$ of relations over D such that e_1 and e_2 evaluate to the same relation on the augmented database $(B, X_1, \dots, X_p)$.

Asking whether there exists a solution of a relational algebra equation on a database is almost exactly the same thing as asking whether an existential second-order logic sentence is true on that database. Hence, by Fagin's theorem [Fag74, EF95], the problems that can be formulated as finding a solution of some relational algebra equation are nothing but the problems in NP.

However, in the present paper, we start from the observation that the set of all solutions of an equation, being a set of tuples of relations, is a *nested* relation. One can therefore consider the enumeration of all solutions of an equation as a query language primitive, which can be added to the nested relational algebra. We introduce and study this extension of the nested relational algebra, which we call the *equation algebra.* The equation algebra is extremely powerful: it is equivalent to the well-known powerset algebra for nested relations. Our particular interest, however, is in what can be expressed in the equation algebra by using only equations that have a solution set of polynomial size on each database. We call such equations *sparse.*

Our interest in sparse equations does not stem from time efficiency considerations. Indeed, it is not obvious how knowing that an equation is sparse would help in actually finding even one solution more quickly. It is neither obvious, however, that it would *not* help. For example, consider the problem of checking on a given database whether some fixed sparse relational algebra equation has a solution. Using an extension of Fagin's theorem to nested relational databases, we show that this problem can be NP-hard only if every problem in NP can already be decided by a polynomial-time non-deterministic Turing machine that has only polynomially many accepting computations on each input. The latter is one of the many unresolved questions in computational complexity theory [All86].

Nevertheless, sparse equations are still interesting from a space efficiency standpoint. Indeed, for the natural evaluation strategy for equation algebra expressions to run in polynomial space, it is necessary that all equations occurring in the expression are sparse. Interest in fragments of powerful query languages for which the natural evaluation strategy is polynomial-space is not new to database theory research. For example, Abiteboul and Vianu [AV91] showed that the parity query is not expressible in the polynomial-space fragment of various computationally complete query languages.

Closer to our topic is the work of Suciu and Paredaens [SP97], who showed that queries such as transitive closure and parity are not expressible in the polynomial-space fragment of the powerset algebra for nested relations.[1] This fragment consists of all powerset algebra expressions where all intermediate results are of polynomial size, on each database. We also mention Grumbach and Vianu [GV95], who also studied a sparsity notion in connection with queries over nested relational databases, although they considered sparsity as a property of databases rather than of query language expressions.

Suciu and Paredaens conjectured in general that the polynomial-space fragment of the powerset algebra has no more power than the nested relational algebra without powerset. (This conjecture has been confirmed for monadic database schemas [VdB].) At first sight, the operator that we add to the nested relational algebra, to enumerate all solutions of an equation, does not seem to be that different from the powerset operator. After all, both operators perform some kind of potentially exponential enumeration.

Yet, as we will point out, the analogue of the Suciu-Paredaens conjecture does not hold for the sparse fragment of the equation algebra. Specifically, using sparse equations only, we can express transitive closure; in fact, we can express any fixpoint query. This complements a result by Abiteboul and Hillebrand [AH95], who showed that transitive closure becomes

[1]This fragment does make sense: there are expressions that always produce a result of logarithmic size. Applying the, exponential, powerset operator to such expressions produces a result of polynomial size.

expressible in the powerset algebra in polynomial space, provided we use a more clever "pipelined" evaluation strategy. Actually, every fixpoint query is already expressible using equations that are not just sparse, but even unambiguous: they have a unique solution on each database. Unambiguous equations in the relational algebra are known as *implicit definitions* in first-order logic, and were studied in the context of finite model theory by Kolaitis [Kol90]. Kolaitis already showed that every fixpoint query can be implicitly defined. We offer a straightforward, direct proof.

Another example is given by the well-known nesting operator of the nested relational algebra. This operator becomes redundant once we extend the algebra with the solution operator or with the powerset operator. However, the original nesting operator never blows up exponentially. We show that, without using the nesting operator itself, nesting is expressible in the equation algebra using sparse equations only, but that the same is not possible with a polynomial-space powerset algebra expression.

However, there are also similarities between the two fragments. Specifically, we prove an analogue to the Suciu-Paredaens result, to the effect that the parity query is not expressible in the sparse fragment of the equation algebra either. This is our main technical contribution. The proof involves an elegant argument, in the style of Liebeck [Lie83], invoking Bochert's theorem on the order of primitive permutation groups.

This paper is organized as follows. Section 2 recalls the nested relational data model. Section 3 introduces relational algebra equations. Section 4 introduces the equation algebra. Section 5 introduces sparse equations, as well as the natural evaluation strategy for equation algebra expressions. Section 6 studies the time complexity of sparse equations. Finally, Section 7 presents the comparison with the polynomial-space powerset algebra.

An appendix with additional details complements the main body of the paper.

2 Preliminaries

We quickly recall the nested relational data model and algebra [TF86, AHV95].

Relation types are defined as follows. The symbol 0 is a type; and, if $\tau_1, \dots, \tau_k$ are types, then so is $(\tau_1, \dots, \tau_k)$. For a type τ, and some set D of atomic values, the *relations of type τ on D* are inductively defined as follows. A relation of type 0 on D is just an element of D (this serves merely as the base case for the induction). A relation of type $(\tau_1, \dots, \tau_k)$ on D is a set of k-tuples $(x_1, \dots, x_k)$ such that x_i is a relation of type τ_i on D, for $i = 1, \dots, k$.

A *database schema* is a finite set $\mathcal{S}$ of relation names, where each relation name has an associated type different from 0. A *database B* over $\mathcal{S}$ consists of a non-empty finite domain D of atomic values, together with, for each relation name R in $\mathcal{S}$, a relation R^B of type τ on D, where τ is the type of R.

The operators of the nested relational algebra are those of the standard relational algebra (union $\cup$ and difference $-$ of relations of the same type; cartesian product $\times$; projection π; selection σ for equality), plus the operators *nesting* ν and *unnesting* μ, defined as follows.

Let R be a relation of type $(\tau_1, \dots, \tau_k)$, and let $i_1, \dots, i_p \in \{1, \dots, k\}$. Then the nesting

$\nu_{i_1,\dots,i_p}(R)$ equals the relation

$$\Big\{\Big(x_1,\dots,x_k,\ \big\{(y_{i_1},\dots,y_{i_p}) \mid (y_1,\dots,y_k) \in R \text{ and } x_j = y_j \text{ for each } j \in \{1,\dots,k\} - \{i_1,\dots,i_p\}\big\}\Big) \ \Big|\ (x_1,\dots,x_k) \in R\Big\}$$

of type $(\tau_1,\dots,\tau_k,(\tau_{i_1},\dots,\tau_{i_p}))$.

Let R be as above, and let $i \in \{1,\dots,k\}$ such that $\tau_i \neq 0$; so τ_i is of the form $(\omega_1,\dots,\omega_\ell)$. Then the unnesting $\mu_i(R)$ equals the relation

$$\{(x_1,\dots,x_k,y_1,\dots,y_\ell) \mid (x_1,\dots,x_k) \in R \text{ and } (y_1,\dots,y_\ell) \in x_i\}$$

of type $(\tau_1,\dots,\tau_k,\omega_1,\dots,\omega_\ell)$.

The expressions of the *nested relational algebra* over a schema $\mathcal{S}$ are now built up using the above operators from the relation names in $\mathcal{S}$ and the symbol D, which stands for the finite domain of the input database. The relation to which an expression e evaluates on a database B is denoted by $e(B)$.

One can extend the nested relational algebra to the *powerset algebra* by adding the powerset operator, defined as follows. Let R be a relation of type $(\tau_1,\dots,\tau_k)$. Then the powerset $\Pi(R)$ equals the relation $\{(S) \mid S \subseteq R\}$ of type $((\tau_1,\dots,\tau_k))$.

3 Equations

Let $\mathcal{S}$ and $\mathcal{X}$ be disjoint database schemas; $\mathcal{S}$ is the actual database schema, while $\mathcal{X}$ is thought of as a set of additional relation variables. Let e_1 and e_2 be two expressions over the expanded schema $\mathcal{S} \cup \mathcal{X}$. We define

Definition 3.1. Given a database B over $\mathcal{S}$, a *solution to the equation* $e_1 = e_2$ is a database A over $\mathcal{X}$ with the same finite domain as B, such that $e_1(B,A) = e_2(B,A)$.

Here, (B,A) denotes the expansion of B with A, i.e., the database over $\mathcal{S} \cup \mathcal{X}$ that has the same finite domain as B, that equals B on $\mathcal{S}$, and that equals A on $\mathcal{X}$.

Example 3.2. For a very simple example, let $R \in \mathcal{S}$ and let $\mathcal{X} = \{X\}$, where X has the same type as R. Then $X \cup R = R$ is an equation. Given a database B over $\mathcal{S}$, a database A over $\{X\}$ is a solution if and only if $X^A \subseteq R^B$.

For another example, let X be a relation variable of type $(0,0)$. One can write a relational algebra expression e such that on any database A over $\{X\}$ with finite domain D, $e(A)$ is empty if and only if X^A is one-to-one, the projections $\pi_1(X^A)$ and $\pi_2(X^A)$ are disjoint, and their union equals D. Then the equation $e = \emptyset$ has a solution on a database B with finite domain D if and only if the cardinality of D is even. (Technically, $e = \emptyset$ is not an equation because the symbol $\emptyset$ is not an expression, but we can easily take $\emptyset$ here to stand for some arbitrary expression that always evaluates to the empty relation.) $\square$

Remark 3.3. In the above example, we used an equation of the special form $e = \emptyset$. Actually, this form is not so special at all, because any equation $e_1 = e_2$ can be brought in this form as $e_1 \,\Delta\, e_2 = \emptyset$, where $e_1 \,\Delta\, e_2$ stands for $(e_1 - e_2) \cup (e_2 - e_1)$ (symmetric difference).

Alternatively, one might wonder about the use of *dis*equations, of the form $e \neq \emptyset$. These are nothing but equations in disguise, because they can also be written as $\pi_1(D \times e) = D$. Conversely, any equation $e_1 = e_2$ can also be written as the disequation $D - \pi_1(D \times (e_1 \Delta e_2)) \neq \emptyset$.

4 The equation algebra

We are now ready to extend the nested relational algebra with a solution operator for equations. We refer to the resulting algebra as the *equation algebra.*

To allow for an elegant definition, we do not fix a schema $\mathcal{S}$ in advance. Rather, we assume a sufficiently large supply of relation names of all possible types. Any relation name can now occur in an expression. Like in logic formulas, some will occur *free* and others will occur *bound.* Bound relation names are bound by the solution of an equation, and serve as the variables of the equation. Within the equation, however, they are still free. We denote the set of relation names that occur free in an equation algebra expression e by $\mathit{free}(e)$.

For the constructs of the nested relational algebra, this is all straightforward: for a relation name R, we have $\mathit{free}(R) := \{R\}$; for expressions e of the form $(e_1 \cup e_2)$, $(e_1 - e_2)$, or $(e_1 \times e_2)$, we have $\mathit{free}(e) := \mathit{free}(e_1) \cup \mathit{free}(e_2)$; for expressions e of the form $\sigma(e')$, $\pi(e')$, $\nu(e')$, or $\mu(e')$, we have $\mathit{free}(e) := \mathit{free}(e')$. For the expression D, we have $\mathit{free}(D) := \emptyset$.

The definition of the new solution operator is now the following:

Definition 4.1. Let e_1 and e_2 be expressions, and let $X_1, \dots, X_p$ be a sequence of distinct relation names. Then

$$\{(X_1, \dots, X_p) \mid e_1 = e_2\}$$

is also an expression. We define its *free* set as $(\mathit{free}(e_1) \cup \mathit{free}(e_2)) - \{X_1, \dots, X_p\}$. We say that the X_i *become bound.*

Note that this is a recursive definition, in the sense that e_1 and e_2 can contain solution operators in turn. To avoid clutter, we disallow equation algebra expressions in which a free relation name at the same time becomes bound in some subexpression, as in $X \times \{(X) \mid X \cup R = R\}$.

An expression e in the equation algebra can be evaluated on databases B over any schema that contains $\mathit{free}(e)$. We already know how this evaluation is defined for the constructs of the nested relational algebra. So we only have to define

Definition 4.2. For a solution expression, e, of the form $\{(X_1, \dots, X_p) \mid e_1 = e_2\}$, and a database B, the evaluation $e(B)$ equals the relation

$$\begin{aligned}&\{(X_1^A, \dots, X_p^A) \mid \\ &\qquad A \text{ is a database over } \{X_1, \dots, X_p\} \\ &\qquad \text{and is a solution of } e_1 = e_2 \text{, given } B\}.\end{aligned}$$

This relation is of type $(\tau_1, \dots, \tau_p)$, where τ_i is the type of X_i for $i = 1, \dots, p$.

Example 4.3. Recall the simple example equation $X \cup R = R$ from Example 3.2. We can turn this equation in the following equation algebra expression e: $\{(X) \mid X \cup R = R\}$. Note that $\mathit{free}(e) = \{R\}$. On any database B over $\{R\}$, the relation $e(B)$ equals $\Pi(R^B)$ (recall the powerset operator Π from Section 2). In other words, the equation algebra expression e is equivalent to the powerset algebra expression $\Pi(R)$.

The equation algebra allows equations to be used inside equations. For example, if we want to compute the powerset of the powerset of R, we can write:

$$\big\{(Y) \mid Y \subseteq \{(X) \mid X \subseteq R\}\big\}.$$

As a third example, let R and T be relation names of the same binary type (τ_1, τ_2). One can write a relational algebra expression e_{tc} such that on any database C over $\{R, T\}$, e_{tc} is empty if and only if $R^C \subseteq T^C$ and T^C is transitively closed. One can also write a nested relational algebra expression $e_{\min}$ that selects, out of a set of binary relations, the minimal ones w.r.t. set inclusion. Then the following equation algebra expression computes the transitive closure of relation R:

$$\pi_{2,3}\mu_1 e_{\min}\big(\{(T) \mid e_{\text{tc}} = \emptyset\}\big). \quad \square$$

In the above example we saw that the powerset operator is expressible in the equation algebra. Conversely, the solution operator is easily expressed in the powerset algebra. Hence,

Proposition 4.4. *The equation algebra is equivalent to the powerset algebra.*

5 Sparse equations

So far, the equation algebra is merely another syntax for the powerset algebra, or, if you want, higher-order logic. However, when we consider a natural evaluation strategy for equation algebra expressions, we start to notice some differences.

By the natural strategy to evaluate a solution expression of the form $\{(X_1, \dots, X_p) \mid e_1 = e_2\}$, we mean the following. Enumerate all databases A over $\{X_1, \dots, X_p\}$, on the finite domain of the given input database, *one by one, reusing the same space.* For each A we test whether it is a solution (by recursively evaluating e_1 and e_2), and if so, we include it in the result.

For the constructs of the nested relational algebra, the natural evaluation strategy is clear: if we have to evaluate an expression of the form $e_1 \Theta e_2$, with $\Theta \in \{\cup, -, \times\}$, we create two intermediate results by recursively evaluating e_1 and e_2, and then apply Θ to these two intermediate results. Similarly, if we have to evaluate an expression of the form $\theta(e)$, with $\theta \in \{\pi, \sigma, \nu, \mu\}$, we create an intermediate result by recursively evaluating e, and then apply θ to this intermediate result.

In view of this natural evaluation strategy, we now define

Definition 5.1. An equation is called *sparse* if all its relation variables are of *flat* type, i.e., of type of the form $(0, \dots, 0)$, and the number of solutions on any given database is at most polynomial in the size of that database.

Example 5.2. The two equations from Example 3.2 are not sparse. Probably the simplest example of a non-trivial sparse equation is the following. Let X be a relation name of type (0). One can write a relational algebra expression e over $\{X\}$ such that on any database A over $\{X\}$, $e(A)$ is empty if and only if X^A is a singleton. Then the equation $e = \emptyset$, where X is taken as the relation variable to be solved for, is sparse. Indeed, given any database B with finite domain D, the solutions are precisely all singleton subsets of D. There clearly are only a linear (and thus at most polynomial) number of possible solutions. $\square$

Remark 5.3. One may wonder whether the notion of sparsity would change if, in Definition 5.1, we would look only at databases over the schema consisting of the relation names that actually occur free in the equation. The answer is easily seen to be negative.

Sparse equations are connected to the natural evaluation strategy in the following way:

Proposition 5.4. *The natural strategy to evaluate an equation algebra expression e runs in polynomial space, if and only if all equations occurring in e are sparse.*

Here, we count not only the space occupied by the intermediate results stored during evaluation, but also the size of the final result. The if-direction of this Proposition is clear; the only-if direction is proven in the Appendix.

6 Time complexity of equation nonemptiness

The *time* complexity of solving sparse equations is closely linked to an open question from computational complexity theory. Unlike the previous section, in this section we are not talking about the natural evaluation strategy, whose time complexity is clearly at least exponential as soon as there are equations to be solved.

Instead, we will be looking at the time complexity of the *nonemptiness problems* of equations. The nonemptiness problem of an equation over a schema $\mathcal{S}$ with relation variables $\mathcal{X}$ is the problem of deciding, given a database over $\mathcal{S}$, whether the equation has a solution on that database. *In the present section we will only consider equations that do not contain equations inside.*

Let us begin by considering equations that are not necessarily sparse, but that still have only flat variables. The nonemptiness problem of such a flat-variable equation is clearly in NP. Now suppose, moreover, that the database schema $\mathcal{S}$ is also flat; then $\mathcal{S} \cup \mathcal{X}$ (the expansion of $\mathcal{S}$ with the relation variables of the equation) is an entirely flat schema. Of course, the equation $e_1 = e_2$ is still in general in the *nested* relational algebra, i.e., e_1 and e_2 can contain ν and μ operators. A result by Paredaens and Van Gucht [PVG92], however, implies that the nested relational algebra condition $e_1 = e_2$ can also be expressed in the form $e \neq \emptyset$, with e a *flat* relational algebra expression. The nonemptiness problem of the equation thus amounts to asking whether $\{(X_1, \ldots, X_p) \mid e \neq \emptyset\}$ on a given database B over $\mathcal{S}$. Equivalently, we ask whether the existential second-order logic ($\exists$SO) sentence $\exists X_1 \ldots \exists X_p \, \varphi_e$ is true on B, where φ_e is a first-order logic sentence expressing that $e \neq \emptyset$. Moreover, by the equivalence of relational algebra and first-order logic, *any* $\exists$SO property can be obtained in this way. Now, Fagin's theorem [Fag74, EF95] states that $\exists$SO captures exactly the NP properties of flat relational databases. Hence, the class of nonemptiness problems of flat-variable equations over flat database schemas is exactly the class of NP properties of flat relational databases.

What if $\mathcal{S}$ is not necessarily flat? We next show that we still get exactly NP. In essence, this is an extension of Fagin's theorem to nested relational databases.

Proposition 6.1. *Every property of nested relational databases over some fixed schema $\mathcal{S}$, which is in NP and closed under isomorphism, corresponds to the nonemptiness problem of some flat-variable equation over $\mathcal{S}$.*

The crux of the proof of this Proposition, given in the Appendix, is a representation of nested relational databases by "pseudo-flat" ones, also used by Gyssens, Suciu, and Van Gucht [GSVG95].

We are now ready to turn to sparse equations. Their nonemptiness problem is not just in NP, but actually in the complexity class *FewP* [All86], consisting of all problems that can be decided by a polynomial-time non-deterministic Turing machine that has at most polynomially many accepting computations on each input. Clearly, P $\subseteq$ FewP $\subseteq$ NP, but the strictness of these inclusions remains open.

The obvious question to ask is whether Proposition 6.1 remains true if we focus on sparse equations, and replace 'NP' by 'FewP.' The answer is an easy "yes," but then we must restrict attention to *ordered* databases: databases that include a total order on their finite domain as one of their relations. Indeed, the usual proof of Fagin's theorem yields the FewP version of that theorem, restricted to ordered databases, for free. This is for flat databases; for general nested relational databases we use the same representation technique as in the proof of Proposition 6.1. As a corollary, we get

Corollary 6.2. *There exists a sparse equation whose nonemptiness problem is NP-complete, if and only if* FewP = NP.

7 Sparse equations versus sparse powerset expressions

Naturally, we call an equation algebra expression sparse if all equations occurring in it are sparse, cf. Proposition 5.4. Inspired by that Proposition, we can also define a sparsity condition on powerset algebra expressions: call a powerset algebra expression *sparse* if its natural evaluation strategy (defined in the obvious way) runs in polynomial space.

Remark 7.1. Using standard techniques one can show that sparsity is undecidable, for equation algebra expressions as wel as powerset algebra expressions. Related questions are also typically undecidable, for example, testing whether a given expression is equivalent to a sparse one.

Suciu and Paredaens [SP97] showed that transitive closure of a flat binary relation is not expressible by a sparse powerset expression. In Example 4.3, we gave an obvious equation algebra expression for transitive closure, but that expression was not sparse. We can do better:

Proposition 7.2. *Transitive closure of a flat relation is expressible by a sparse equation algebra expression.*

Proof. Given a binary relation R and a natural number $n \geq 1$, we define the relation R^n as $R \circ \cdots \circ R$ (n times R), where $\circ$ is the classical composition operator of binary relations: $S \circ T = \pi_{1,4}\sigma_{2=3}(S \times T)$. Further, define $R^{\leq n}$ as $\bigcup_{i=1}^{n} R^i$, and define $R^{=n}$ as $R^n - R^{\leq n-1}$. Note that $R^{\leq |R|}$ equals the transitive closure of R, and that for $n > |R|$, $R^{\leq n} = R^{\leq |R|}$.

Now consider the following 6-ary relation *Run*:

$$\mathit{Run} := \bigcup_{i=1}^{|R|} R^{\leq i} \times R^{\leq i+1} \times R^{=i+1}.$$

In the Appendix, we show that there is an equation whose *only* solution, given R, is *Run*. This proves the Proposition, because all we then have to do is unnest the solution set and project on the middle two columns to get the transitive closure. (The only exception is when *Run* is empty, in which case the transitive closure of R is R itself, but this can also easily be tested in the nested relational algebra.) $\square$

Remark 7.3. The equation constructed in the above proof is not only sparse, it is *unambiguous:* it has a unique solution on each input database. Moreover, the same proof works more generally for any *fixpoint query* [AHV95] on flat databases. The only difficulty is that fixpoint queries start from the empty relation, while in our proof of Proposition 7.2 we start from R, but that is easily dealt with. As already explained in the Introduction, we thus basically rediscovered an earlier result by Kolaitis to the effect that every fixpoint query is implicitly definable in first-order logic [Kol90]. But note the directness of our proof, straightforwardly specifying the run of the fixpoint computation in an unambiguous way. The original proof (also presented by Ebbinghaus and Flum [EF95]) is a bit more roundabout, specifying the "stage comparison" relation instead.

Another, perhaps a bit frivolous, example of a query that is expressible using sparse equations but not using sparse powerset expressions is the nesting operator ν. It is easy to express ν in the powerset algebra using the powerset operator and the other operators, but not ν itself; so ν is not primitive in the powerset algebra. As a consequence (Proposition 4.4), ν is not primitive in the equation algebra either. We next observe that when we restrict to sparse expressions, nesting remains imprimitive in the equation algebra, but becomes primitive again in the powerset algebra.

Proposition 7.4. *Nesting is not expressible by a sparse powerset expression without using the ν operator itself.*

The proof of this Proposition, given in the Appendix, is a standard argument in finite model theory, involving quantifier elimination on monadic structures.

Proposition 7.5. *Nesting is expressible by a sparse equation expression without using the ν operator itself.*

Proof. Let R be a relation name of type $(0,0)$, and let X and Y be relation variables of type (0). We can write a relational algebra expression e such that on any database C over $\{R, X, Y\}$, $e(C)$ is empty if and only if X is a singleton $\{x\}$ with $x \in \pi_1(R)$, and $Y = \{y \mid (x,y) \in R\}$. Hence, the expression

$$\mu_1\big(\{(X,Y) \mid e = \emptyset\}\big)$$

is a sparse equation expression equivalent to $\nu_2(R)$.

The construction for general nesting operations is analogous. □

Our final, and main technical, contribution concerns the parity query. Suciu and Paredaens showed that the parity of the cardinality of a finite set is not expressible by a sparse powerset expression. We show the analogue for the equation algebra:

Proposition 7.6. *The parity query is not expressible by a sparse equation expression.*

Proof. Suppose, to the contrary, that we have a sparse equation expression to express the parity of the cardinality of a finite domain D. We may assume that the input schema is empty, i.e., an input database consists of D and nothing else. Consider the innermost equation occurring in our expression. As in Section 6, it can be written in the form $\{(X_1, \dots, X_p) \mid e \neq \emptyset\}$, where e is a flat relational algebra expression over the flat schema $\{X_1, \dots, X_p\}$.

Now let A be a solution, given an input D of size n. Then for every permutation f of D, $f(A)$ is also a solution. The number of different such $f(A)$ is precisely $n!/|\mathrm{Aut}(A)|$, where $\mathrm{Aut}(A)$ is the group of automorphisms of A. Since the equation is supposed to be sparse, this number is at most n^ℓ for some fixed ℓ, or, equivalently, $|\mathrm{Aut}(A)| \geq n!/n^\ell$. Putting $k = \ell + 1$, this implies $|\mathrm{Aut}(A)| \geq (n-k)!$ for sufficiently large n.

We thus need to know more about large permutation groups. The following crucial lemma, proved in the Appendix, will give us the information we need. The group of permutations of a finite set D is denoted by $\mathrm{Sym}(D)$, and its alternating subgroup of even permutations by $\mathrm{Alt}(D)$. If G is a subgroup of $\mathrm{Sym}(D)$, a *fixed set* for G is a subset $\Delta \subseteq D$, such that every $g \in G$ maps Δ to Δ. The action of G on a fixed set Δ (as a subgroup of $\mathrm{Sym}(\Delta)$) is denoted by G^Δ.

Lemma 7.7. *Let k be a fixed natural number. Let G be a subgroup of* $\mathrm{Sym}(D)$, *$|D| = n$, n sufficiently large. Then $|G| \geq (n-k)!$ implies the existence of a fixed set Δ with $|\Delta| = n-k$, such that G^Δ contains* $\mathrm{Alt}(\Delta)$.

Invoking this Lemma for $G = \mathrm{Aut}(A)$, we get a fixed set Δ of size $n-k$ such that any even permutation of Δ can be extended to an automorphism of A.

Now let X be one of the relation variables of the equation, of arity, say, r, and consider any r-tuple t whose components are either the symbol $*$ or are in $D - \Delta$. Let r' be the number of components that are the symbol $*$. Further, let ξ be an equality type of r'-tuples. Denote by $\mathit{Neighbors}_X^A(t,\xi)$ the set of r'-tuples over Δ of equality type ξ such that, if we replace the $*$-components of t by the components of the r'-tuple (from left to right), we get a tuple in A^X. The following claim is proven in the Appendix:

Claim 7.8. *$\mathit{Neighbors}_X^A(t,\xi)$ is either empty, or consists of* all *r'-tuples over Δ of equality type ξ.*

We call (t,ξ) an *r-ary pattern.* If $\mathit{Neighbors}_X^A(t,\xi)$ is nonempty (and thus full), we say that the pattern is *instantiated in X^A*. Note also that the extreme cases, where t consists exclusively of stars or where t has no star at all, are also allowed and make sense.

By the above, we thus see that any solution A can be generated by the following non-deterministic procedure:

1. Initialize all relations of A to empty.

2. Choose k different elements from D, playing the role of the elements outside Δ.

3. For every relation variable X (of arity r, say), run through all r-ary patterns, and for each of them, non-deterministically instantiate it in X^A, or not.

Since k and $X_1, \ldots, X_p$ are fixed, the number of possible patterns is also fixed. Hence, we can write an expression in the nested relational algebra that, given D, constructs the set of all possible outputs A of the above non-deterministic procedure. This set is a superset of the solution set of the equation. Selecting these solutions using the nested relational algebra is also an easy matter.

Hence, we can get rid of the innermost equation. Repeating this process, we can get rid of all equations, so that in the end we are left with a standard nested relational algebra expression for the parity query. But this is well known to be impossible [AHV95]. $\square$

Acknowledgment

We are indebted to László Babai, who pointed us to Liebeck's paper.

References

[AH95] S. Abiteboul and G. Hillebrand. Space usage in functional query languages. In G. Gottlob and M.Y. Vardi, editors, *Database Theory—ICDT'95*, volume 893 of *Lecture Notes in Computer Science*, pages 439–454. Springer-Verlag, 1995.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[All86] E. Allender. The complexity of sparse sets in P. In A.L. Selman, editor, *Structure in Complexity Theory*, volume 223 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1986.

[AV91] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proceedings 23rd ACM Symposium on the Theory of Computing*, pages 209–219, 1991.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R.M. Karp, editor, *Complexity of Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 43–73. 1974.

[GSVG95] 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, 1995. Full version to appear in *Information and Computation.*

[GV95] S. Grumbach and V. Vianu. Tractable query languages for complex object databases. *Journal of Computer and System Sciences*, 51(2):149–167, 1995.

[Kol90] Ph.G. Kolaitis. Implicit definability on finite structures and unambiguous computations. In *Proceedings 5th IEEE Symposium on Logic in Computer Science*, pages 160–180, 1990.

[Lie83] M.W. Liebeck. On graphs whose full automorphism group is an alternating group or a finite classical group. *Proc. London Math. Soc.*, 47(2):337–362, 1983.

[PVG92] J. Paredaens and D. Van Gucht. Converting nested algebra expressions into flat algebra expressions. *ACM Transactions on Database Systems*, 17(1):65–93, 1992.

[SP97] 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.

[TF86] S. Thomas and P. Fischer. Nested relational structures. In P. Kanellakis, editor, *The Theory of Databases*, pages 269–307. JAI Press, 1986.

[VdB] J. Van den Bussche. Simulation of the nested relational algebra by the flat relational algebra, with an application to the complexity of evaluating powerset algebra expressions. *Theoretical Computer Science.* To appear.

[Wie64] H. Wielandt. *Finite Permutation Groups.* Academic Press, 1964.

Appendix

Proof of Proposition 5.4. The if-direction is clear. For the only-if direction, we work by induction on the nesting depth of equations. The base case—expressions that do not contain any equations at all—is trivial.

For the inductive step, consider a top-level equation $\{(X_1, \dots, X_p) \mid e_1 = e_2\}$ occurring in e. The natural strategy to evaluate this equation runs in polynomial space, so in particular, for each expansion of each database B over *free*(e) with a candidate solution A over $\{X_1, \dots, X_p\}$, the natural evaluation of e_1 and e_2 on (B, A) runs in polynomial space. Now all such expansions (B, A) generate exactly all databases over *free*$(e) \cup \{X_1, \dots, X_p\}$, so that we can conclude that the natural evaluations strategies of e_1 and e_2 in general run in polynomial space.[2] Hence, by induction, all equations occurring nested inside a top-level equation are sparse.

[2]Formally, we must note here that e_1 and e_2 might not actually mention certain relation names in *free*(e) or $\{X_1, \dots, X_p\}$, and that there is still the formal possibility that their natural evaluation might not run in polynomial space on databases over schemas not containing these names. However, using Remark 5.3, it can be seen that this is impossible.

The top-level equation itself must also be sparse. Indeed, if one of the X_i would be of non-flat type, even one candidate solution can already be of exponential size, which is impossible. Further, since we store the solution set as an intermediate result, it must be of at most polynomial size on all databases over *free*(e). Since the individual solutions are flat databases and thus of polynomial size, the cardinality of the solution set must therefore be at most polynomial. $\square$

Proof of Proposition 6.1. The crux is a representation of nested relational databases by "pseudo-flat" ones, also used by Gyssens, Suciu, and Van Gucht [GSVG95]. Given a nested relational database B, we define its *extended domain,* denoted by edom(B), as the union of its finite domain of atomic values with the set of all relations occurring (possibly deeply nested) in B. We regard the relations in the extended domain as if they were atomic values. Now for any nested relational database schema $\mathcal{S}$ we can construct a flat one $\bar{\mathcal{S}}$, together with a mapping *rep* from the set of databases over $\bar{\mathcal{S}}$ *onto* the set of databases over $\mathcal{S}$, expressible in the nested relational algebra. The details of this mapping need not concern us here. Important is that we can furthermore construct a converse mapping *flat* from the set of databases over $\mathcal{S}$ to the set of databases over $\bar{\mathcal{S}}$, also expressible in the nested relational algebra, with the following properties for each database B over $\mathcal{S}$: (1) the finite domain of *flat*(B) equals edom(B); and (2) *rep*$(\textit{flat}(B)) = B$. Note that when *flat* is expressed in the nested relational algebra, the result *flat*(B) is not really a flat database, because of the nested relations in the extended domain. However, it is "pseudo-flat," in the sense that we regard these relations as if they were atomic values. For any relation name R of $\bar{\mathcal{S}}$, we denote the nested relational algebra expression defining the R-component of the mapping *flat* by flat_R. Likewise, we denote the expression defining the D-component by flat_D.

Given this representation, the proof is straightforward. Let L be an NP property of databases over $\mathcal{S}$, closed under isomorphism. Define the property $\bar{L}$ of databases over $\bar{\mathcal{S}}$ as follows: F satisfies $\bar{L}$ if $\mathit{rep}(F)$ satisfies L. Then $\bar{L}$ is in NP, and is also closed under isomorphism. Hence, Fagin's theorem gives us an $\exists$SO sentence $\exists X_1 \dots \exists X_p \, \varphi$ over $\bar{\mathcal{S}}$ expressing $\bar{L}$. By the equivalence of relational algebra and first-order logic, there is a flat relational algebra expression e over $\bar{\mathcal{S}} \cup \{X_1, \dots, X_p\}$ such that the first-order logic sentence φ is equivalent to $e \neq \emptyset$. Now modify e as follows: for every relation name R of $\bar{\mathcal{S}}$, replace every occurrence of R in e by flat_R. Likewise, replace every occurrence of D in e by flat_D. Denote the resulting nested relational algebra expression by e'.

We now have, for any database B over $\mathcal{S}$, that B satisfies L if and only if $\exists X_1 \dots X_p \, e' \neq \emptyset$ is true on B. The condition $e' \neq \emptyset$ can easily be written as an equation (cf. Remark 3.3). $\square$

Proof of Proposition 7.2 (continued). The desired equation expresses the conjunction of the following conditions on relation variable X:

1. For any pair $(x_5, x_6) \in \pi_{5,6}(X)$, we denote the relation
$$\{(x_1, x_2, x_3, x_4) \mid (x_1, \dots, x_6) \in X\}$$
by $\tilde{X}(x_5, x_6)$, and denote further
$$\hat{X}(x_5, x_6) := \pi_{1,2}(\tilde{X}(x_5, x_6)) \quad \text{and}$$
$$\hat{\hat{X}}(x_5, x_6) := \pi_{3,4}(\tilde{X}(x_5, x_6)).$$
Then for every $(x, y) \in \pi_{5,6}(X)$, we must have
$$\tilde{X}(x,y) = \hat{X}(x,y) \times \hat{\hat{X}}(x,y);$$
$$\hat{X}(x,y) \supseteq R;$$
$$\hat{\hat{X}}(x,y) = \hat{X}(x,y) \cup \hat{X}(x,y) \circ R;$$
and
$$(x,y) \in \hat{\hat{X}}(x,y) - \hat{X}(x,y).$$
Furthermore, *every* pair (x', y') in the latter difference belongs to $\pi_{5,6}(X)$, with $\hat{X}(x', y') = \hat{X}(x,y)$ (and thus also $\hat{\hat{X}}(x', y') = \hat{\hat{X}}(x,y)$).
2. $R^{=2} \subseteq \pi_{5,6}(X)$, and for every $(x,y) \in R^{=2}$, we have $\hat{X}(x,y) = R$.
3. For every $(x,y) \in \pi_{5,6}(X)$ such that $\hat{\hat{X}}(x,y) \circ R - \hat{\hat{X}}(x,y) \neq \emptyset$, there exists a pair $(x', y') \in \pi_{5,6}(X)$ with $\hat{X}(x', y') = \hat{\hat{X}}(x,y)$.
4. For every $(x,y) \in \pi_{5,6}(X)$ such that $\hat{X}(x,y) \neq R$, there exists a pair $(x', y') \in \pi_{5,6}(X)$ with $\hat{\hat{X}}(x', y') = \hat{X}(x,y)$. $\square$

Proof of Proposition 7.4. Suppose we want to express nesting of a flat binary relation R. The first application of the powerset operator is to the result of a flat relational algebra expression e applied to R. Let us focus on the case where R is the identity relation on a finite domain D of n elements, where n is sufficiently large. By a standard quantifier elimination argument, we can see that either $e(R)$ is empty on all such R, or $e(R)$ is of size at least n. In the first case, the powerset operator is useless, and there must be another powerset operator in the overall expression, to which we apply the same argument. In the second case, the powerset operator explodes and the overall expression is not sparse. $\square$

Proof of Claim 7.8. Suppose to the contrary that $N := \mathit{Neighbors}_X^A(t, \xi)$ is neither empty nor full. Take h_1 in N, and take h_2 (of arity r' and of type ξ) not in N. Take two arbitrary elements from Δ that neither appear in h_1 nor in h_2, and remove them from Δ, resulting in Δ'. Take a third tuple h_3 (of the right arity and type) over Δ', and disjoint from h_1 and h_2. If h_3 is in N, initialize the set I to $\{h_2, h_3\}$; otherwise, put $I := \{h_1, h_3\}$. Now complete I to a maximal set of pairwise disjoint r'-tuples over Δ' of equality type ξ. There are at least $(n-k-2)/r'$ tuples in I.

Assume at least half of I is outside N; denote the set of these by I'. (The case where at least half of I is *in* N is symmetric.) Fix an $h \in I \cap N$. For each tuple s in I', we consider the permutation ϖ_s that transposes s and h and leaves everything else fixed. If ϖ_s happens to be odd, we make it even by adding the transposition of the two dummy elements we took out of Δ (when we defined Δ'). Then each set $\varpi_s(N)$ contains s, but does not contain any other tuples from I'. Thus, we produce in this way at least $f(n) := ((n-k-2)/2r'$ different sets of r'-tuples over Δ. Since they are even, each ϖ_s can be extended to an automorphism. Hence, each of the $f(n)$ sets must be the $\mathit{Neighbors}_X^A(t', \xi)$ of some t'. However, there are less than $(k+1)^r$ different possibilities for t', while $f(n)$ is larger than that for n sufficiently large. So we get to the desired contradiction. □

Proof of Lemma 7.7. For background on finite permutation groups, we refer to Wielandt's book [Wie64], but here are a few preliminaries. An *orbit* of a permutation group G on a set D is a set of the form $\{g(x) \mid g \in G\}$, for some $x \in D$. We call G *transitive* if D is one single orbit of G. Further, G is called *primitive* if it is transitive and has no nontrivial blocks. Here, a *block* of G is a subset $\Delta \subseteq D$ such that for all $g \in G$, the set $g(\Delta)$ is either equal to Δ, or disjoint from it. *Trivial* blocks are $\emptyset$, D, and the singletons. If G is not primitive but transitive, there is always a *complete block system* which partions D in equal-sized blocks. We recall:

Bochert's Theorem (1889). *Let G be primitive on D, not containing* $\mathrm{Alt}(D)$. *Let $|D| = n$. Then $|G| \leq n!/\lceil n/2 \rceil!$.*

For the proof of the Lemma, first assume that G is transitive. There are two possibilities:

1. G is imprimitive with, say, b blocks of size a ($a > 1$, $b > 1$, $ab = n$). Then $|G| \leq b!\,(a!)^b$, which in turn is at most $2(\lfloor n/2 \rfloor !)^2$ for n sufficiently large. Thus, by what is given about $|G|$,
$$(n-k)! \leq 2(\lfloor n/2 \rfloor !)^2. \tag{1}$$
However, this is impossible for n sufficiently large.

2. G is primitive. Then, unless G contains $\mathrm{Alt}(D)$ (in which case the Lemma is proved), by Bochert's theorem,
$$(n-k)! \leq \frac{n!}{\lceil n/2 \rceil !}. \tag{2}$$
Again, this is impossible for n sufficiently large.

Now assume G is intransitive. Let Δ be an orbit of G of maximal size; let $\ell \geq 1$ be such that the size of Δ equals $n - \ell$. We have $|G| \leq (n-\ell)!\,\ell!$. Suppose $\ell > k$. Then $(n-\ell)!\,\ell!$ reaches its maximum at $\ell = k+1$. Hence, $(n-k)! \leq |G| \leq (n-k-1)!\,(k+1)!$ and thus $n - k \leq (k+1)!$ which is impossible for large enough n.

So, $\ell \leq k$, or in other words, the size of Δ is at least $n-k$. We have $|G| \leq \ell!\,|G^{\Delta}| \leq k!\,|G^{\Delta}|$, so $|G^{\Delta}| \geq (n-k)!/k!$. By definition G^{Δ} is transitive on Δ. We can now apply the same arguments as in the case "G is transitive" above, for G^{Δ} instead of G, and get that G^{Δ} must contain $\mathrm{Alt}(\Delta)$. Indeed, in the right-hand sides of inequalities 1 and 2, n now becomes $n - \ell$, which has for effect that the upper bounds become smaller. Hence, if these inequalities were impossible (and they were), they now become even more impossible. The extra factor of $1/k!$ in the left-hand sides does not have a significant influence. $\square$