On the inversion of relational queries

Joachim Biskup Jan Paredaens Jan Van den Bussche

February 18, 1998

1 Introduction

A user of a relational database often does not see the actual relations of the database, but rather sees a *view* of that database. This view is the result of some query applied to the database. Now suppose our user wants to derive information about the database from the view he sees. Thereto, assuming he knows the exact query the result of which he is seeing, he may try to compute the inverse image of that query on his view.

In this paper, we initiate a theoretical study of the inversion of relational queries.

First, we look at the complexity of inverting queries expressed in the relational algebra. We will show that this issue is closely connected with known results and questions from computational complexity theory.

Then, we consider inversion of queries as a query language construct that can be added to existing query languages. Thereto we will work in the *nested* relational database model, since queries are in general not injective and thus the inverse image of a query in general consists of a collection of relations; such a collection can be naturally represented as a nested relation.

2 Preliminaries

In this section we recall the basic definitions concerning relational databases and the relational algebra. We will assume a certain background in database theory from the reader. A modern treatment of the subject is given in the book by Abiteboul, Hull and Vianu [2]; background on any unexplained notion or fact which we will use can be found there.

Assume a sufficiently large supply of *relation names* is given, where each relation name has an associated *arity* (a natural number). A *database schema* is a finite set of relation names.

Assume further a domain V of data values is given. Let $\mathcal{S}$ be a database schema. A *database over* $\mathcal{S}$ is a mapping on $\mathcal{S}$ which assigns to each relation name $R \in \mathcal{S}$ a finite n-ary relation on V, where n is the arity of R.

We note:

Proviso *Whenever we write 'relation', we will always mean a finite relation.*

Now fix a schema $\mathcal{S}$. The set $\mathcal{A}$ of *relational algebra expressions over* $\mathcal{S}$ is defined inductively as follows.

- Each relation name $R \in \mathcal{S}$ is in $\mathcal{A}$.
- If e_1 and e_2 are in $\mathcal{A}$, of the same arity n, then $(e_1 \cup e_2)$ and $(e_1 - e_2)$ are in $\mathcal{A}$, both also of arity n.
- If e_1 and e_2 are in $\mathcal{A}$, of arities n_1 and n_2 respectively, then $(e_1 \times e_2)$ is in $\mathcal{A}$, of arity $n_1 + n_2$.
- If e is in $\mathcal{A}$, of arity n, then
  - $\sigma_{i=j}(e)$ is in $\mathcal{A}$, also of arity n, where $i, j \in \{1, \ldots, n\}$;
  - $\pi_{i_1,\ldots,i_p}(e)$ is in $\mathcal{A}$, of arity p, where $i_1, \ldots, i_p \in \{1, \ldots, n\}$.

Let d be a database over $\mathcal{S}$. For a relational algebra expression e over $\mathcal{S}$, we define $e(d)$, the *result of evaluating* e *on* d, as follows.

- $R(d) := d(R)$, for $R \in \mathcal{S}$.
- $(e_1 \cup e_2)(d) := e_1(d) \cup e_2(d)$ and $(e_1 - e_2)(d) := e_1(d) - e_1(d)$.
- $(e_1 \times e_2)(d) := \{(x_1, \ldots, x_{n_1}, y_1, \ldots, y_{n_2}) \mid (x_1, \ldots, x_{n_1}) \in e_1(d)$ and $(y_1, \ldots, y_{n_2}) \in e_2(d)\}$.
- $\sigma_{i=j}(e)(d) := \{t \in e(d) \mid t_i = t_j\}$.
- $\pi_{i_1,\ldots,i_p}(e)(d) := \{(t(i_1), \ldots, t(i_p)) \mid t \in e(d)\}$.

So, an expression of arity n defines a mapping from databases to n-ary relations. Such a mapping is called a *query*.

3 Complexity of inverting relational algebra queries

Fix a database schema $\mathcal{S}$. Assume we are given a relational algebra expression e over $\mathcal{S}$, and a relation r. Suppose we want to compute the full inverse image of e on r, i.e., all databases d over $\mathcal{S}$ such that $e(d) = r$. This is clearly impractical, as there could easily be infinitely many such databases.

One way to restrict the possible databases to a finite search space is based on the notion of *active domain*. The active domain of a relation r (or database d), denoted by $\text{adom}(r)$ (or $\text{adom}(d)$), is the set of all data values appearing in r (or d). The active domain is always finite, and the number of databases with some specified active domain is also finite. We now define:

Definition 1 Let e be an expression and r a relation. Then $e^{-1}(r)$ is defined as the set of databases $\{d \text{ over } \mathcal{S} \mid \text{adom}(d) \subseteq \text{adom}(r) \text{ and } e(d) = r\}$.

It may well be still impractical to compute e^{-1} as just defined, if only because the output is already exponential in size. We call the fortunate cases where this does not happen "polysize-invertible". Formally, define the *size* of a relation r (or database d) simply as the cardinality of its active domain.[1] Then we define:

Definition 2 An expression e is called *polysize-invertible* if for each r, the cardinality of $e^{-1}(r)$ is at most some fixed polynomial in the size of r. Expression e is called *1-invertible* if for each r, the cardinality of $e^{-1}(r)$ is at most 1.

In order to get a feeling for the notions just introduced, in the following example we check them for the individual operators of the relational algebra.

Example 1

- Let $\mathcal{S} = \{R_1, R_2\}$ and $e = R_1$. Then e is clearly not polysize-invertible unless the arity of R_2 is 0. Of course, if $\mathcal{S}$ consists of R_1 only, e is 1-invertible.
- Again with $\mathcal{S} = \{R_1, R_2\}$, consider $e = R_1 \cup R_2$. Since for any relation r, $r \cup s = r$ for any subset s of r, e is not polysize-invertible (again unless the arity is 0).

[1] All reasonable definitions of size will be polynomially equivalent with the one we use here.

- Now let $e = R_1 - R_2$. If the arity is 1, for any relation r the only database d for which $e(d) = r$ is $(r, \emptyset)$;[2] hence, in this case e is 1-invertible. However, if the arity is 2 or more, e is not. Indeed, suppose for concreteness that the arity is 2. Then for any relation r containing only identical pairs, we have $(s \cup r) - s = r$ for any subset s of the difference relation on adom(r).

- Still with $\mathcal{S} = \{R_1, R_2\}$, the expression $R_1 \times R_2$ is 1-invertible. Indeed, $r_1 \times r_2 = r$ necessarily implies that r_1 equals the projection of r on its first n_1 columns and r_2 equals the projection on its last n_2 columns (where n_1 and n_2 are the arities of R_1 and R_2, respectively).

- Let $\mathcal{S} = \{R\}$ and $e = \sigma_{i=j}(R)$. Since for any relation r and for any relation s such that $\sigma_{i=j}(s)$ is empty, $\sigma_{i=j}(r \cup s) = \sigma_{i=j}(r)$, e is not polysize-invertible.

- Finally, let $e = \pi_{i_1,\ldots,i_p}(R)$. Then again, unless $\{i_1, \ldots, i_p\} = \{1, \ldots, n\}$, where n is the arity of R, e is not polysize-invertible. ■

The reader will have noticed that in those cases of the example where we had polysize-invertibility, we actually had 1-invertibility. Of course 1-invertibility implies polysize-invertibility, but the converse does not hold. For example, let $\mathcal{S} = \{R_1, R_2\}$ where R_1 is unary and R_2 is binary. One can write an expression e defining the query "if relation R_2 consists of exactly one pair, of the form (a, a), where a is in relation R_1, then return relation R_1; else return $\emptyset$". This e is clearly not 1-invertible but it is polysize-invertible.

Also, if an expression e is injective then it is certainly 1-invertible. Again the converse does not hold. For example, take $\mathcal{S}$ consisting of a single relation name R. One can write an expression e defining the query "if relation R contains at least three tuples, return relation R itself; else return $\emptyset$". This e is clearly not injective but it is 1-invertible.

A natural question that arises is the following: does polysize-invertibility of an expression e guarantee that computing e^{-1} is tractable? And more generally, what exactly is the computational complexity of inverting relational

[2] If the relation names $R_1, \ldots, R_m$ of a schema $\mathcal{S}$ are listed in some order that is clear from the context, we will use $(r_1, \ldots, r_n)$ to denote the database d over $\mathcal{S}$ with $d(R_1) = r_1$, $\ldots$, $d(R_m) = r_m$.

algebra queries? In order to investigate this, for the remainder of this section, we will view e^{-1} as a decision problem: *given r, is $e^{-1}(r)$ non-empty?*

Observe that this decision problem is in NP. Actually the converse holds as well:

Proposition 1 *Every property $\mathcal{P}$ of relations that is in NP and that is closed under isomorphism can be expressed as the inversion of a relational algebra query.*

Proof. This is nothing but Fagin's theorem [6, 5]. Indeed, by this theorem, for any such property $\mathcal{P}$ there is a first-order logic sentence $\varphi(R_1, \ldots, R_m, R)$ over the relation symbols R_1, ..., R_m and R, such that relation r satisfies property $\mathcal{P}$ iff there exist relations $r_1, \ldots, r_m$ on adom(r) such that $\varphi(r_1, \ldots, r_m, r)$ is true.[3] Since the relational algebra is equivalent to first-order logic, we can write an expression e over the schema $\{R_1, \ldots, R_m, R\}$ that defines the query "on input database $(r_1, \ldots, r_m, r)$, if $\varphi(r_1, \ldots, r_m, r)$ is true then return r; else return $\emptyset$". Clearly, for this e and non-empty r, $e^{-1}(r)$ is non-empty iff r satisfies $\mathcal{P}$. ■

Let us now make the question raised above a little bit more specific: is there a polysize-invertible expression e such that e^{-1} is NP-complete? We next link this problem with a known open problem from computational complexity theory. A problem is in the complexity class *FewP* (introduced by Allender [4]) if it can be decided by a non-deterministic polynomial-time Turing machine which has at most polynomially many accepting computations on each input. Clearly, P $\subseteq$ FewP $\subseteq$ NP, but no inclusion is known to be strict. We have:

Theorem 1 *There is a polysize-invertible expression e such that e^{-1} is NP-complete, if and only if* FewP = NP.

Proof. The only-if implication is immediate: if e is polysize-invertible then e^{-1} is clearly in FewP. So we have an NP-complete problem in FewP and thus FewP = NP.

For the converse implication, assume FewP = NP. Take an arbitrary expression e_0 such that e_0^{-1} is NP-complete. By assumption there is a FewP Turing machine M deciding $e_0^{-1}(r) \neq \emptyset$ for any input relation r.

[3] This is with the understanding that, when giving relation r as input to, say, a Turing machine, adom(r) is identified with an initial segment of the natural numbers.

Now we need to recall the general idea of the usual proof of Fagin's theorem. To test whether r is accepted by an NP Turing machine such as M, one uses a first-order sentence $\varphi(R_1, \ldots, R_m, R)$ that says that relation R_1 is a linear order on the active domain of relation R, and that the remaining relations $R_2, \ldots, R_m$ encode an accepting computation of M on input relation R, where the active domain is presented on the input tape in the order specified by R_1. Note that the relational algebra expression e corresponding to φ (in the way described a few paragraphs earlier) will be "almost", but not quite, polysize-invertible. Indeed, since we know that the number of accepting computations of M is polynomially bounded, the only reason why e is not polysize-invertible is because there are an exponential number of possible orders of the active domain. Hence, if we can eliminate the need for an explicit order relation R_1 in φ we are done. This is possible as follows.

For notational simplicity assume the arity of R is 2. For a binary relation r and a linear order $<$ on $\mathrm{adom}(r)$, define the 4-ary relation $r_<$ as follows:

$$r_< := (< \times \{(a,a) \mid a \in \mathrm{adom}(r)\}) \cup (r \times \{(a,b) \mid a, b \in \mathrm{adom}(r) \text{ and } a \neq b\}).$$

Let R' be a new relation name of arity 4. Now write an expression e' over the schema $\{R_2, \ldots, R_m, R'\}$ defining the following query: "on input database $(r_2, \ldots, r_m, r')$, if r' is of the form $r_<$ and $\varphi(<, r_2, \ldots, r_m, r)$ holds, return r'; else return $\emptyset$". This e' is polysize-invertible. The set of positive elements of the decision problem for e'^{-1} equals $\{r_< \mid e_0^{-1}(r) \neq \emptyset \text{ and } < \text{ is a linear order on } \mathrm{adom}(r)\}$. Since the NP-complete decision problem for e_0^{-1} is easily reducible to the decision problem for e'^{-1}, the latter is NP-complete as well. Hence the theorem is proven. ■

4 Preliminaries on nested relations

Each component of a tuple in a relation of a relational database is an "atomic" data value. The *nested* relational database model, originally introduced by Thomas and Fischer [13] and Schek and Scholl [11], provides for relations where tuple components can also be relations in turn. In this section we briefly recall this model.

Again we assume a sufficiently large supply of relation names, but now each relation name has an associated *type*, rather than an arity. Types are inductively defined as follows:

- The symbol 0 is a type.
- If $\tau_1, \ldots, \tau_k$ are types, then so is $(\tau_1, \ldots, \tau_k)$.

We assume the type of a relation name is never 0; the type 0 will only be used to get an elegant inductive definition in the next paragraph.

Recall that V is our domain of data values. We define the *relations of type* τ, for a type τ, inductively as follows:

- A relation of type 0 is an element of V.
- A relation of type $(\tau_1, \ldots, \tau_k)$ is a finite set of k-tuples $(x_1, \ldots, x_k)$, such that x_i is a relation of type τ_i.

The elements of a relation of type τ are also called *tuples of type* τ.

A *database schema* is, as before, a finite set $\mathcal{S}$ of relation names. A *database over* $\mathcal{S}$ is then a mapping on $\mathcal{S}$, assigning to each relation name R of $\mathcal{S}$ a relation of type τ, where τ is the type of R.

Notice that a relation of type $(0, \ldots, 0)$ (n zeros) is a classical n-ary relation on V; we will call such relations *flat*.

We can generalize the relational algebra to work on nested relational databases by canonically generalizing the five operators of the relational algebra and adding two new ones: *nesting* (ν) and *unnesting* (μ). Fix a schema $\mathcal{S}$. The set $\mathcal{NA}$ of *nested relational algebra expressions over* $\mathcal{S}$ is defined inductively as follows:[4]

- Each relation name $R \in \mathcal{S}$ is in $\mathcal{NA}$.
- If e_1 and e_2 are in $\mathcal{NA}$, of the same type τ, then $(e_1 \cup e_2)$ and $(e_1 - e_2)$ are in $\mathcal{NA}$, both also of type τ.
- If e_1 and e_2 are in $\mathcal{NA}$, of types $(\tau_1, \ldots, \tau_k)$ and $(\omega_1, \ldots, \omega_\ell)$ respectively, then $(e_1 \times e_2)$ is in $\mathcal{NA}$, of type $(\tau_1, \ldots, \tau_k, \omega_1, \ldots, \omega_\ell)$.
- If e is in $\mathcal{NA}$, of type $\tau = (\tau_1, \ldots, \tau_n)$, then
  - $\sigma_{i=j}(e)$ is in $\mathcal{NA}$, also of type τ, where $i, j \in \{1, \ldots, n\}$.

[4]The nested relational algebra we are going to define is a slight variation of the original one by Thomas and Fischer. The expressive power is exactly the same.

- $\pi_{i_1,\ldots,i_p}(e)$ and $\nu_{i_1,\ldots,i_p}(e)$ are in $\mathcal{NA}$, where $i_1,\ldots,i_p \in \{1,\ldots,n\}$; $\pi_{i_1,\ldots,i_p}(e)$ is of type $(\tau_{i_1},\ldots,\tau_{i_p})$, and $\nu_{i_1,\ldots,i_p}(e)$ is of type $\tau \cdot ((\tau_{i_1},\ldots,\tau_{i_p}))$.[5]
- $\mu_i(e)$ is in $\mathcal{NA}$, where $i \in \{1,\ldots,n\}$ and $\tau_i \neq 0$; it is of type $\tau \cdot (\omega_1,\ldots,\omega_\ell)$, where $\tau_i = (\omega_1,\ldots,\omega_\ell)$.

Let d be a database over $\mathcal{S}$. For a nested relational algebra expression e over $\mathcal{S}$, we define $e(d)$, the *result of evaluating e on d*, as follows. All cases, except nesting and unnesting, are verbatim the same as in the ordinary relational algebra. So we only give the two remaining cases:

- $\nu_{i_1,\ldots,i_p}(e)(d) := \{t \cdot (s) \mid t \in e(d) \text{ and } s = \{(t'_{i_1},\ldots,t'_{i_p}) \mid t' \in e(d) \text{ and } t \text{ and } t' \text{ agree on all components different from } i_1,\ldots,i_p\}\}$.
- $\mu_i(e)(d) := \{t \cdot (x_1,\ldots,x_\ell) \mid t \in e(d) \text{ and } (x_1,\ldots,x_\ell) \in t_i\}$.

So, an $\mathcal{NA}$-expression e of type τ defines a mapping (query) from databases to relations of type τ.

Excellent illustrations of how one actually uses $\mathcal{NA}$ to express a variety of queries can be found in the papers by Gyssens and Van Gucht [9, 8]. In the sequel, we will almost never substantiate claims of the kind that a certain query is expressible in $\mathcal{NA}$, as these claims will always be obvious to the reader familiar with $\mathcal{NA}$ or similar formalisms.

5 Extending the nested relational algebra with inversion

In Section 3, we considered the inversion of relational queries in isolation. In this section, we introduce inversion as a query language construct. This allows us to compute the inverse image of some expression on the result of some other expression, or to apply some expression to the result of inverting some other expression. Since the result of an inversion is a set of databases, we will work in the nested relational model, where such sets are easily representable as nested relations.

[5] The dot denotes concatenation of two tuples.

Apart from the supply of relation names (recall the previous section), we assume a similar supply of relation *variables*. Relation variables are entirely similar to relation names; in particular, they also have an associated type.

The set $\mathcal{NA}^{-1}$ of *nested algebra expressions with inversion* is defined by extending the formation rules of $\mathcal{NA}$ with one new base case and one new construction. The second base case, besides that stating that every relation name is an expression in $\mathcal{NA}$, states:

- *Every relation variable is in* $\mathcal{NA}^{-1}$.

Each expression e of $\mathcal{NA}^{-1}$ also has an associated set $\text{free}(e)$ of *free relation variables*. For R a relation name, $\text{free}(R) := \emptyset$; for X a relation variable, $\text{free}(X) := \{X\}$. For expressions e of the form $(e_1 \cup e_2)$, $(e_1 - e_2)$ or $(e_1 \times e_2)$, $\text{free}(e) := \text{free}(e_1) \cup \text{free}(e_2)$. For expressions e of the form $\sigma_{\ldots}(e')$, $\pi_{\ldots}(e')$, $\nu_{\ldots}(e')$ or $\mu_{\ldots}(e')$, $\text{free}(e) := \text{free}(e')$.

We are now ready to introduce the new construction of $\mathcal{NA}^{-1}$:

- *Let* e_1 *and* e_2 *be in* $\mathcal{NA}^{-1}$*, both of type* τ*. Let* $\text{free}(e_2) = \{X_1, \ldots, X_m\}$*, where* X_i *is of type* τ_i*. Then* $(e_2)^{-1}(e_1)$ *is in* $\mathcal{NA}^{-1}$*, of type* $(\tau_1, \ldots, \tau_m)$*. The set of free variables of this expression equals* $\text{free}(e_1)$.

An expression e in $\mathcal{NA}^{-1}$ with free variables is evaluated on databases d over the augmented schema $\mathcal{S} \cup \text{free}(e)$. For the formation rules of $\mathcal{NA}$, the definition of $e(d)$ is verbatim the same as for $\mathcal{NA}$. For the two formation rules we have added, we have:

- $X(d) := d(X)$, for a relation variable X.
- $(e_2)^{-1}(e_1)(d) := \{(d'(X_1), \ldots, d'(X_m)) \mid d'$ a database over $\mathcal{S} \cup \{X_1, \ldots, X_m\}$ such that $d'|_{\mathcal{S}} = d|_{\mathcal{S}}$, $\text{adom}(d') \subseteq \text{adom}(e_1(d))$, and $e_2(d') = e_1(d)\}$.

Of course, at the end we are only interested in $\mathcal{NA}^{-1}$-expressions without free variables (which may however contain subexpressions *with* free variables).

At this point an example is in order:

Example 2 Take a schema $\mathcal{S}$, and a relation name $R \in \mathcal{S}$, of type τ.

1. Let X be a relation variable, also of type τ. Consider the expression $e = (X)^{-1}(R)$ of type (τ). Note that $\text{free}(e)$ is empty. Given a database d over $\mathcal{S}$, $e(d)$ equals the singleton relation $\{(d(R))\}$.

2. Let X and Y be relation variables of type τ. One can write an $\mathcal{NA}$-expression e over $\{X, Y\}$ expressing the query "if relation X is a subset of relation Y, return relation Y; else return $\emptyset$". Then the expression $\pi_1(e)^{-1}(R)$ returns on each database d over $\mathcal{S}$ the powerset of $d(R)$, i.e., the relation $\{(s) \mid s \subseteq d(R)\}$ of type (τ). ■

The second example above shows that the powerset operator is expressible in $\mathcal{NA}^{-1}$. The extension of the nested relational algebra with the powerset has been well investigated [1, 9, 8]. It is easy to express inversion in the powerset algebra. Hence we have:

Proposition 2 *$\mathcal{NA}^{-1}$ is equivalent in expressive power with the powerset algebra.*

The same example also implies the following:

Proposition 3 *The fragment of $\mathcal{NA}^{-1}$ that allows only inversion expressions $(e_2)^{-1}(e_1)$ where e_2 is an $\mathcal{NA}$-expression (i.e., does not contain inversions in turn) is equivalent to the full $\mathcal{NA}^{-1}$.*

Indeed, the inversion used to express powerset is in that fragment.

6 Complexity considerations in $\mathcal{NA}^{-1}$

We would like to generalize the property of polysize-invertibility, considered in Section 3, to a property of general $\mathcal{NA}^{-1}$-expressions. This property, which we will call simply *polysize*, will say that in the evaluation of the $\mathcal{NA}^{-1}$-expression, only intermediate results of polynomial size will appear.

In order to define polysize expressions formally, we first need to define the *size* of a nested relation formally. We do this inductively as follows:

- The size of a data value is 1.
- The size of a tuple is 1 plus the sum of the sizes of its components.
- The size of a relation is 1 plus the sum of the sizes of its elements.

The size of a database is then simply the sum of the sizes of its relations.

We next need a formal definition of the *intermediate results* in the evaluation of an $\mathcal{NA}^{-1}$-expression e on a database d, denoted by $\text{int}(e, d)$. We do this inductively as follows:

- $\text{int}(R, d) := \{d(R)\}$, for a relation name R.
- $\text{int}(X, d) := \{d(X)\}$, for a relation variable X.
- $\text{int}(e_1 \,\Theta\, e_2, d) := \text{int}(e_1, d) \cup \text{int}(e_2, d) \cup \{(e_1 \,\Theta\, e_2)(d)\}$, for $\Theta = \cup$, $-$, or $\times$.
- $\text{int}(\theta(e)) := \text{int}(e, d) \cup \{\theta(e)(d)\}$, for $\theta = \sigma_{\ldots}$, $\pi_{\ldots}$, $\nu_{\ldots}$, or $\mu_{\ldots}$.
- $\text{int}((e_2)^{-1}(e_1), d) := \text{int}(e_1, d) \cup \bigcup\{\text{int}(e_2, d') \mid d'$ a database over $\mathcal{S} \cup \{X_1, \ldots, X_m\}$ such that $d'|_{\mathcal{S}} = d|_{\mathcal{S}}$ and $\text{adom}(d') \subseteq \text{adom}(e_1(d))\}$.

We now define:

Definition 3 Expression e is said to be *polysize* if for each database d and each relation $r \in \text{int}(e, d)$, the size of r is at most some fixed polynomial in the size of d.

Intuitively, e is polysize if the *natural evaluation strategy* of e can be implemented to run in polynomial space. We will not define this natural evaluation strategy formally; it suffices to say that it naturally follows the inductive definition of expression evaluation, where in the case of inversion it tries all possible databases d', using the same space for different possibilities. Note that $\mathcal{NA}^{-1}$-expressions without inversion (i.e., $\mathcal{NA}$-expressions) are always polysize.

In the remainder of this section we illustrate the concept of polysize by means of three case studies: *nesting*, *transitive closure*, and *parity*.

Nesting. In $\mathcal{NA}^{-1}$ there are two constructs that increase the depth of relation types: the nesting operator ν, and inversion. The natural question arises whether or not one can be simulated by the other (and the remaining constructs of $\mathcal{NA}^{-1}$), or, equivalently, whether both are *primitive* within $\mathcal{NA}^{-1}$. One part of this question is readily answered: inversion is clearly primitive within $\mathcal{NA}^{-1}$, since we mentioned above that $\mathcal{NA}^{-1}$ is equivalent with the powerset algebra which is well-known to be much more powerful than the nested relational algebra.

The other part of the question is also readily answered: nesting is well-known not to be primitive within the powerset algebra. Let us illustrate this

by an example. Suppose we want to compute $\nu_2(r)$, where r is the following relation of type $(0,0)$:

$$r = \begin{array}{|cc|}\hline a & b \\ a & c \\ b & c \\ \hline \end{array}$$

We begin by computing $r_1 := \Pi(\pi_2(r))$, where Π denotes the powerset operator:

$$r_1 = \begin{array}{|c|}\hline \quad \\ \hline b \\ \hline c \\ \hline \begin{array}{|c|}\hline b \\ c \\ \hline \end{array} \\ \hline \end{array}$$

We then compute $r_2 := \{(x,s) \mid (x) \in \pi_1(r) \text{ and } s = \{(y) \mid (x,y) \in r\}\}$:

$$r_2 := \begin{array}{|cc|}\hline a & \begin{array}{|c|}\hline b \\ c \\ \hline \end{array} \\ b & \begin{array}{|c|}\hline c \\ \hline \end{array} \\ \hline \end{array}$$

The desired result $\nu_2(r)$ now simply equals $\pi_{1,2,4}\sigma_{1=3}(r \times r_2)$.

This way of computing nesting does not yield an expression in $\mathcal{NA}^{-1}$ that is polysize, since the application of the powerset operator produces an intermediate result of exponential size. We next show that we can do better:

Proposition 4 *There is a polysize $\mathcal{NA}^{-1}$-expression that defines nesting without using the ν operator.*

Proof. Let R be a relation name of type $(0,0)$; we show how to express $\nu_2(R)$. The argument for general relation types is notationally more complicated but entirely analogous.

Let X, Y and Z be relation variables of types (0), (0) and $(0,0)$, respectively. One can write an $\mathcal{NA}$-expression e over $\{X,Y,Z\}$ defining the following query: "if X consists of one single tuple (x), and $(x) \notin \pi_1(Z)$, then return $(X \times Y) \cup Z$; else return $\emptyset$". If $e(r_1,r_2,r_3) = r$ then $r_1 \times r_2$ is an equivalence class of tuples in r under the equivalence "agreeing on the first component"; r_3 then is the union of all other equivalence classes. The number of such equivalence classes is linear in the cardinality of $\pi_1(r)$. Hence, $(e)^{-1}(R)$ is polysize. Moreover, we have the following equivalence:

$$\nu_2(R) \equiv \pi_{1,2,4}\sigma_{1=3}(R \times \pi_{4,2}\mu_1(e)^{-1}(R)).$$ ■

Example 3 Let us illustrate the proof on the same example relation r as above. We first compute $r'_1 := (e)^{-1}(r)$:

$$r'_1 = \left\{ \begin{array}{|c|c|cc|} \hline \boxed{a} & \boxed{\begin{array}{c} b \\ c \end{array}} & \boxed{b} & \boxed{c} \\ \boxed{b} & \boxed{c} & \boxed{\begin{array}{cc} a & b \\ a & c \end{array}} & \\ \hline \end{array} \right\}$$

We then compute $\pi_{4,2}\mu_1(r'_1)$, and see that this yields exactly relation r_2 from the previous example. ■

We next indicate that although exactly the same queries are expressible $\mathcal{NA}^{-1}$ and in the powerset algebra, the two formalisms nevertheless differ in terms of the complexity of their natural evaluation strategies. Indeed, in contrast to the previous proposition, we have:

Proposition 5 *There is no polysize expression in the powerset algebra that defines nesting without using the ν operator.*

Proof. Let R be a relation name of type $(0,0)$. Let e be a powerset algebra expression e over $\{R\}$ that is equivalent to $\nu_2(R)$ and that does not use ν.

Expression e must contain at least one application of the powerset operator, since this is the only operator, besides ν itself, that increases the depth of relation types. Let e' be the first subexpression of e to which a powerset operator is applied, in the natural evaluation strategy of e. So e' is a flat relational algebra expression; it can be equivalently expressed by a first-order logic formula φ over $\{R\}$, where the quantifiers in φ are understood to range only over the active domain of the input relation.

Now consider all relations r_n of the form:

$$\left\{ \begin{array}{|cc|} \hline 1 & 1 \\ 2 & 2 \\ \hline \end{array} \right\}$$
$$\vdots$$
$$\left\{ \begin{array}{|cc|} \hline n & n \\ \hline \end{array} \right\}$$

By a standard inductive argument [5], it can be shown that for large enough n, φ is equivalent, on all relations r_n, to a quantifier-free formula in the language of equality. Furthermore, it is readily verified that any such formula

defines either the empty relation on every r_n, or a relation of size at least n. In the second case, the powerset of $e'(r_n)$ will be of exponential size and hence e is not polysize. In the first case, e' is useless on all inputs r_n with large enough n and hence e will contain at least one other application of the powerset, on which we can repeat the same reasoning. ■

Transitive closure. A much more difficult to prove result of the same nature as Proposition 5 was obtained by Suciu and Paredaens [12]. Their result does not concern nesting, but *transitive closure*. The transitive closure query maps each relation of type $(0,0)$ to its transitive closure. Suciu and Paredaens showed that no polysize expression in the powerset algebra can define this query. Transitive closure is well-known to be expressible in the powerset algebra, so the adjective 'polysize' is crucial here. (Transitive closure is not expressible in $\mathcal{NA}$.) It remains open whether the analogue of Proposition 4 for transitive closure holds: *is there a polysize $\mathcal{NA}^{-1}$-expression that defines transitive closure?*

While we do not have the answer to this question, we offer another observation on the expressibility of transitive closure in $\mathcal{NA}^{-1}$. The obvious way of computing the transitive closure of a relation r in the powerset algebra is to produce the powerset of $\mathrm{adom}(r) \times \mathrm{adom}(r)$, from which we then select the minimal relation containing r that is transitively closed. (Since the powerset is an intermediate result, this does not yield a polysize expression.) An alternative (actually, computationally much more expensive) way, however, is to produce all possible finite sequences of relations with active domain $\mathrm{adom}(r)$, from which we then select the sequence that starts with r and is generated by repeatedly applying the relational algebra expression $R \cup \pi_{1,4}\sigma_{2=3}(R \times R)$. Here, sequences of relations are represented as relations of type $((0,0),(0,0))$.

We can nicely express this way of computing transitive closure in $\mathcal{NA}^{-1}$ as follows. Let X be a relation variable of type $((0,0),(0,0))$. One can write an $\mathcal{NA}$-expression e defining the following query over $\{X, R\}$: "if

1. there is a tuple t in relation X such that $t(1)$ equals relation R;
2. for every tuple t in relation X we have $t(2) = t(1) \cup \pi_{1,4}\sigma_{2=3}(t(1) \times t(1))$; and
3. for every tuple t in relation X there is another tuple t' in X such that $t'(1) = t(2)$,

then return relation R; else return $\emptyset$". Then for any non-empty relation r, $e^{-1}(r)$ will be a singleton $\{s\}$ containing the sequence computing the transitive closure of r. From this s we can easily extract the second component of the unique tuple t in s with $t(1) = t(2)$; this gives us the desired result.

Note that, although the expression e above could be called "1-invertible" in the sense of Definition 2, the $\mathcal{NA}^{-1}$-expression $(e)^{-1}(R)$ is *not* polysize, since for any relation r the size of one single relation X of type $((0,0),(0,0))$ whose active domain is contained in $\mathrm{adom}(r)$ can already be exponential in the size of r. Hence, the just described way of expressing transitive closure in $\mathcal{NA}^{-1}$ is really more a curiosity than anything else.

Parity. Another typical query not expressible in $\mathcal{NA}$ but expressible in the powerset algebra is *parity*, which maps each relation r of type (0) to r itself if its cardinality is even, and to $\emptyset$ otherwise. We can ask the same questions we asked for nesting and for transitive closure: is there a polysize expression defining parity in the powerset algebra? or in $\mathcal{NA}^{-1}$? Paredaens and Suciu's proof techniques apply also to parity; so again the answer for the powerset algebra is negative. Also, the question for $\mathcal{NA}^{-1}$ again remains open.

Note that parity is particularly elegantly (though not polysize) expressible in $\mathcal{NA}^{-1}$ as follows. Let X be a relation variable of type $(0,0)$. One can write a relational algebra expression e over $\{X,R\}$ defining the following query: "if relation X is a bijection with $\pi_1(X) \cup \pi_2(X) = R$ and $\pi_1(X) \cap \pi_2(X) = \emptyset$, then return X; else return $\emptyset$". The parity query can be expressed simply as "if $(e)^{-1}(R) \neq \emptyset$, return R; else return $\emptyset$".

At this point one may wonder whether there exist queries at all that are not expressible in $\mathcal{NA}$ but expressible by a polysize $\mathcal{NA}^{-1}$-expression. The example we just gave for parity can be adapted, using a standard "padding" technique, to answer this question affirmatively:

Proposition 6 *The polysize fragment of $\mathcal{NA}^{-1}$ is strictly more powerful than $\mathcal{NA}$.*

Proof. Let R be a relation name of type $((0,0))$. Consider the following query over $\{R\}$: "if relation R holds all permutations of its active domain, and the cardinality of the active domain is even, return R; else return $\emptyset$." This query is not expressible in $\mathcal{NA}$. To see how the query can be polysize expressed in $\mathcal{NA}^{-1}$, let X be a relation variable of type $(0,0)$. One can write an $\mathcal{NA}$-expression e over $\{X,R\}$ defining the following query: "if relation

X is a bijection, $\pi_1(X) \cup \pi_2(X)$ equals the active domain of relation R, and $\pi_1(X) \cap \pi_2(X) = \emptyset$, return R; else return $\emptyset$". One can also write an $\mathcal{NA}$-expression e' over $\{R\}$ defining the following query: "if relation R holds all permutations of its active domain, return R; else return $\emptyset$". Then $(e)^{-1}(e')$ is polysize. Indeed, if $e'(r) \neq \emptyset$, then the size of r will be of the order of $n!$, where n is the cardinality of the active domain of r. Now the number of bijections from one half of an n-element set to another half is less than $n!$. The query under consideration is hence expressed in polysize $\mathcal{NA}^{-1}$ as "if $(e)^{-1}(e'(R)) \neq \emptyset$, return R; else return $\emptyset$", or formally,

$$\pi_1(R \times (e)^{-1}(e'(R))).$$ ■

7 Decidability issues

We conclude this paper by considering some decidability issues. A first observation is that the properties of expressions discussed above, such as polysize-invertibility of a relational algebra expression, or polysizeness of powerset algebra or $\mathcal{NA}^{-1}$ expressions, are clearly undecidable. We will prove only one case, the other ones being analogous.

Proposition 7 *The problem whether a given relational algebra expression is polysize-invertible, is undecidable.*

Proof. Let R and S be relation names of arity 2. It is well known that the problem whether a given relational algebra expression e over $\{R\}$ evaluates to empty on all databases, is undecidable. Now for any such e, one can write an expression e' over $\{R, S\}$ defining the following query: "if $e = \emptyset$ return $\emptyset$; else return the active domain of relation S." Then e' is polysize-invertible if and only if e evaluates to empty on all databases, which is undecidable. Hence, the problem under consideration is undecidable as well. ■

Given the fact that strictly more queries can be expressed in $\mathcal{NA}^{-1}$ than in the nested relational algebra, the following proposition shows that the associated natural problem is undecidable:

Proposition 8 *The problem whether a given $\mathcal{NA}^{-1}$-expression is equivalent to a nested relational algebra expression, is undecidable.*

Proof. Let us restrict attention to expressions defining flat queries, and show that already in this case the problem is undecidable. Nested relational algebra expressions defining flat queries are always equivalent to flat relational algebra expressions [10]. The problem whether a given Datalog program is bounded is undecidable [7]. Moreover, a Datalog program is bounded if and only if it is equivalent to a relational algebra expression [3]. Now Datalog programs can be expressed in the powerset algebra. Hence, the problem whether a given powerset algebra expression (defining a flat query) is equivalent to a relational algebra expression, is undecidable. Because powerset algebra expressions can be translated into equivalent $\mathcal{NA}^{-1}$-expressions (Proposition 2), the corresponding problem for $\mathcal{NA}^{-1}$-expressions is also undecidable, which we had to prove. ■

Finding restrictions on the expressions, or on the allowed databases, or both, such that the problems considered above become decidable, is an interesting topic for further research.

References

[1] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. *The VLDB Journal*, 4(4):727–794, 1995. Originally INRIA Research Report 846, 1988.

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

[3] M. Ajtai and Y. Gurevich. Datalog vs first-order logic. *Journal of Computer and System Sciences*, 49(3):562–588, 1994.

[4] 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.

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

[6] 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.

[7] H. Gaifman, H. Mairson, Y. Sagiv, and M.Y. Vardi. Undecidable optimization problems for database logic programs. *Journal of the ACM*, 40(3):683–713, 1993.

[8] M. Gyssens and D. Van Gucht. A comparison between algebraic query languages for flat and nested databases. *Theoretical Computer Science*, 87(2), 1991.

[9] M. Gyssens and D. Van Gucht. The powerset algebra as a natural tool to handle nested database relations. *Journal of Computer and System Sciences*, 45(1):76–103, 1992.

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

[11] H.-J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[12] D. Suciu and J. Paredaens. Any algorithm in the complex object algebra with powerset needs exponential space to compute transitive closure. In *Proceedings 13th ACM Symposium on Principles of Database Systems*, pages 201–209. ACM Press, 1994.

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