On the Immanence of relational genius

Joachim Giesing Jan Wendlandt Jan Van Den Bussche

February 10, [illegible]

1 Introduction

We view of a relational database often been not see the actual relations of the database, but rather view a view of that database. This view is the result of some query applied to the database. Now suppose one more wants to derive information about the database from the view he view. Obviously, assuming he knows the most query the result of which he is seeing, he may try to compute the known image of that query on his view.

In this paper, we initiate a theoretical study of the immanence of relational genius.

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

Then, we consider immanence of genius as a query language constructed that can be added to existing query languages. Obviously we will work in the main relational database model, since genius are in general not injective and thus the known 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. We refer interested of the reader to given in the book by Abiteboul, Hull and Vianu [3]. Background on any complexity notion or fact which we will use can be found there.

Assume a sufficiently large supply of sel flos a are in place, allow each selection mean flos an associated alg [a natural number]. W [illegible] is a finite set of selection means.

Assume further a Boolean [illegible] of data values is given. Let [illegible] be a Boolean values. W [illegible] is a mapping on [illegible] which assigns to each selection means [illegible] a finite n-ary selection on [illegible], where n is the arity of [illegible].

The notion

Boolean [illegible] a site 'sel flos', a [illegible] [illegible] a finite sel flos.

Now for a subset [illegible]. The set [illegible] of sel flos [illegible] applications [illegible] is defined inductively as follows.

- Each selection means [illegible] is in [illegible].
- If s_1 and s_2 are in [illegible], of the same arity n, then $\{s_1 \cap s_2\}$ and $\{s_1 - s_2\}$ are in [illegible], that also of arity n.
- If s_1 and s_2 are in [illegible], of arities n_1 and n_2 respectively, then $\{s_1 \times s_2\}$ is in [illegible], of arity $n_1 + n_2$.
- If s is in [illegible], of arity n, then
  - $m_{i \to j}\{s\}$ is in [illegible], also of arity n, where $i, j \in \{1, \ldots, n\}$;
  - $m_{i_1, \ldots, i_n}\{s\}$ is in [illegible], of arity n, where $i_1, \ldots, i_n \in \{1, \ldots, n\}$.

Let U be a Boolean [illegible]. For a selectional algebra expression s [illegible], [illegible] $s\{U\}$, the result of [illegible] s [illegible] U, as follows.

- $[illegible]\{U\} := U\{[illegible]\}$, for [illegible].
- $\{s_1 \cap s_2\}\{U\} := s_1\{U\} \cap s_2\{U\}$ and $\{s_1 - s_2\}\{U\} := s_1\{U\} - s_2\{U\}$.
- $\{s_1 \times s_2\}\{U\} := \{\{w_1, \ldots, w_{n_1}, u_1, \ldots, u_{n_2}\} \mid \{w_1, \ldots, w_{n_1}\} \in s_1\{U\}$ and $\{u_1, \ldots, u_{n_2}\} \in s_2\{U\}\}$.
- $m_{i \to j}\{s\}\{U\} := \{t \in s\{U\} \mid t_i = t_j\}$.
- $m_{i_1, \ldots, i_n}\{s\}\{U\} := \{\{t(i_1), \ldots, t(i_n)\} \mid t \in s\{U\}\}$.

So, an expression of arity n defines a mapping from Boolean to n-ary selections. Such a mapping is called a [illegible].

1 Complexity of in-order relational algebra queries

We a database volume $\mathbb{D}$. Assume we are given a relational algebra expression e over $\mathbb{D}$, and a relation s. Suppose we want to compute the full inverse image of s on e, i.e., all databases D over $\mathbb{D}$ such that $e[D] = s$. This is clearly impractical, as there could really 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 s for database D, denoted by $\mathrm{adom}(s)$ for $\mathrm{adom}(D)$, is the set of all data values appearing in s for D. The active domain is always finite, and the number of databases with same specified active domain is also finite. We now define:

Definition 1 Let e be an expression and s a relation. Then $e^{-1}[s]$ is defined as the set of databases $\{D$ over $\mathbb{D} \mid \mathrm{adom}(D) \subseteq \mathrm{adom}(s)$ and $e[D] = s\}$.

It may well be still impractical to compute e^{-1} as just defined, if only because the output is clearly exponential in size. We call the fundamental issue this does not happen "polynomial-invertible". Formally, define the size of a relation s for database D simply as the cardinality of its active domain.[1] Then we define:

Definition 2 An expression e is called polynomial-invertible if for each s, the cardinality of $e^{-1}[s]$ is at most some fixed polynomial in the size of s. Expression e is called 1-invertible if for each s, the cardinality of $e^{-1}[s]$ is at most 1.

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

Example 1

- Let $\mathbb{D} = \{\mathbb{D}_1, \mathbb{D}_2\}$ and $e = \mathbb{D}_1$. Then e is clearly not polynomial-invertible unless the arity of $\mathbb{D}_1$ is 1. Of course, if $\mathbb{D}$ consists of $\mathbb{D}_1$ only, e is 1-invertible.

- Again with $\mathbb{D} = \{\mathbb{D}_1, \mathbb{D}_2\}$, consider $e = \mathbb{D}_1 \cup \mathbb{D}_2$. Since for any relation s, $s \cup s = s$ for any subset s of s, e is not polynomial-invertible (again unless the arity is 0).

[1] Well-ordered databases of size $|D|$ be polynomially equivalent to the one e was here.

- Now let $\varphi = M_1 \to M_2$. If the arity is 1, for any valuation v the only formula Φ for which $v[\Phi] = v$ is $[x, \Phi]$.[1] Hence, in this case φ is 1-inventible. However, if the arity is 2 or more, φ is not. Indeed, suppose for contradiction that the arity is 2. Then for any valuation v satisfying only finitely many, we have $[v \models \Phi v] = v$ for any subset v of the finitely valuation on $\mathrm{dom}[v]$.

- Still with $M = [M_1, M_2]$, the composition $M_1 \circ M_2$ is 1-inventible. Indeed, $v_1 \circ v_2 = v$ necessarily implies that v_1 agrees the projection of v on its first v_1 coloring and v_2 agrees the projection on its last v_1 coloring [where v_1 and v_2 are the restrictions of M_1 and M_2, respectively].

- Let $M = [M]$ and $v = v_{1\to 1}[M]$. Since for any valuation v and for any valuation v such that $v_{1\to 1}[v]$ is empty, $v_{1\to 1}[v \models v] = v_{1\to 1}[v]$, v is not polynom-inventible.

- Finally, let $v = v_{1,\ldots,1_n}[M]$. Then again, unless $[\ v, \ldots, \ v] = [1, \ldots, v]$, where v is the arity of M, v is not polynom-inventible. ■

The author will here notice that in these cases of the example where we find polynom-inventibility, we actually find 1-inventibility. Of course 1-inventibility implies polynom-inventibility, but the converse does not hold. For example, let $M = [M_1, M_2]$ where M_1 is among and M_2 is Among. One can write an expression v holding the group "If valuation M_1 results of exactly one pair, of the form $[v, v]$, where v is in valuation M_1, then return valuation M_1 also return Φ". This v is clearly not 1-inventible but it is polynom-inventible.

Also, if an expression v is injective then it is certainly 1-inventible. Again the converse does not hold. For example, take M consisting of a single valuation name M. One can write an expression v holding the group "If valuation M contains at least three tuples, return valuation M [result] also return Φ". This v is clearly not injective but it is 1-inventible.

A natural question that arises is the following: Does polynom-inventibility of an expression v guarantee that computing v^{-1} is tractable? Well more generally, what exactly is the computational complexity of inverting relational

[1] If the relation names $M_1, \ldots, M_m$ of a schema S are listed in some order that is clear from the context, v [[ ]] can [$v_1, \ldots, v_m$] be denote the database S over M [[ ]] $S[M_1] = v_1$, $\ldots$, $S[M_m] = v_m$.

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

Observe that this Boolean problem is in NP. Naturally the converse holds as well:

Proposition 1. Suppose $\mathcal{P}$ is a fixed DFA M is in NP $\Leftrightarrow$ M is also in co-NP. In other words, $\mathcal{P}$ is a fixed DFA $\Leftrightarrow$ the language $\mathcal{P}$ has a co-NP.

Proof. This is nothing but Ladner's theorem [8, 9]. Indeed, by this theorem, for any small property $\mathcal{P}$ there is a first-order logic sentence $\phi(M_1, \ldots, M_m, M)$ over the relation symbols $M_1, \ldots, M_m$ and M, such that relation x satisfies property $\mathcal{P}$ iff there exist relations $x_1, \ldots, x_m$ on $\mathrm{dom}(x)$ such that $\phi(x_1, \ldots, x_m, x)$ is true.[1] Since the relational algebra is equivalent to first-order logic, we can write an expression e over the schema $\{M_1, \ldots, M_m, M\}$ that defines the query "on input database $\{x_1, \ldots, x_m, x\}$, if $\phi(x_1, \ldots, x_m, x)$ is true then return x; else return $\emptyset$". Clearly, for this x and non-empty x, $x^{-1}(x)$ is non-empty iff x satisfies $\mathcal{P}$. ■

Let us now make the question which arises a little bit more specific: is there a polynomial-time expression e such that e^{-1} is NP-complete? We must find this problem with a known open problem from computational complexity theory. W problem is in the complexity class Co NP (introduced by Williams [9]) 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, if NP Conp NP, then no instance is known to be stated. We have:

Theorem 1. There is a polynomial-time expression e such NP e^{-1} is NP-complete, if e only if Conp = NP.

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

For the converse implication, assume Conp = NP. Take an arbitrary expression e_0 such that e_0^{-1} is NP-complete. By assumption there is a Conp Turing machine NP Deciding $e_0^{-1}(x)$ ⇔ ∅ for any input relation x.

[1] This is NP the mathematically that, for given relation x as input to, say, a Turing machine, $\mathrm{dom}(x)$ is identical NP an initial segment of the natural numbers.

Now we need to recall the general form of the usual proof of Rogin's theorem. The test relation r is accepted by an NP Turing machine call it M, and there r that-action instances $g\{M_1, \ldots, M_m, M\}$ that says that relation M_1 is a linear action on the action tableau of relation M, and that the remaining relations $M_2, \ldots, M_m$ encode an accepting computation of M on input relation M, where the action tableau is generated on the input tape in the action specified by M_1. Note that the relational algebra expression e corresponding to g {in the way tableau r has perpendicular action} will be "closed", but not quite, polynomial-invertible. Instead, since we know that the encoding of accepting computations of M is polynomially bounded, the only reason why e is not polynomial-invertible is because there are an exponential number of possible values of the action tableau. Hence, if we can eliminate the need for an explicit action relation M_1 in g we are done. This is possible as follows.

For relational simplicity assume the arity of M is 2. For a fixed relation r and a linear order $<$ on $\mathrm{dom}\{r\}$, define the 2-ary relation $r_<$ as follows:

$$r_< := \{\langle r \times \{\langle v, v\rangle \mid v \in \mathrm{dom}\{r\}\}\rangle \mid$$
$$\in \{r \times \{\langle v, u\rangle \mid v, u \in \mathrm{dom}\{r\} \text{ and } v \neq u\}\}.$$

Let M' be a new relation name of arity 4. Now write an expression e' over the schema $\langle M_1, \ldots, M_m, M'\rangle$ defining the following query: "on input tableaux $\{r_1, \ldots, r_m, r'\}$, if r' is of the form $r_<$ and $g\{M, r_1, \ldots, r_m, r\}$ holds, return r'; else return $\emptyset$". While e' is polynomial-invertible. The set of positive elements of the tableau product for e'^{-1} equals $\{r_< \mid r_<^{-1}\{r\} \neq \emptyset$ and r is a linear order on $\mathrm{dom}\{r\}\}$. Since the NP-complete tableau product for $r_<^{-1}$ is easily reducible to the tableau product for e'^{-1}, the latter is NP-complete as well. Hence the theorem is proven. ∎

8 Conclusions on small relations

Small components of a tuple in a relation of a relational database is an "atomic" data value. The main relational database model, originally introduced by Codd and Klug [10] and Codd and Codd [11], provides for relations whose tuple components are also in relations in turn. In this section we briefly recall this model.

Given an arbitrary or infinitely large supply of relation names, that are small relation names for an associated type, written Dom as follows:

- The symbol $\mathbb{B}$ is a type.
- If $\sigma_1, \ldots, \sigma_n$ are types, then so is $\{\sigma_1, \ldots, \sigma_n\}$.

We assume the type of a relation name is name $\mathbb{R}$; the type $\mathbb{B}$ will only be used to get an elegant inductive definition in the next paragraph.

Recall that $\mathbb{D}$ is some domain of data values. We define the set $\mathrm{Dom}(\sigma)$ for a type σ, for a type σ, inductively as follows:

- A relation of type $\mathbb{B}$ is an element of $\mathbb{D}$.
- A relation of type $\{\sigma_1, \ldots, \sigma_n\}$ is a finite set of n-tuples $\{v_1, \ldots, v_n\}$, such that v_i is a relation of type σ_i.

The elements of a relation of type σ are also called tuples of type σ.

W ! ! a a is, as before, a finite set of relation names. W ! ! a a is then a mapping on , assigning to each relation name of a relation of type σ, where σ is the type of .

Notice that a relation of type $\{\mathbb{B}, \ldots, \mathbb{B}\}$ n times is a classical n-ary relation on $\mathbb{D}$; we will call such relations !.

We now generalize the relational algebra to work on nested relational data-bases by semantically generalizing the five operators of the relational algebra and adding two new ones: nesting $\{n\}$ and unnesting $\{u\}$. We a . The set of such of the ! tuple expressions over is defined inductively as follows:[1]

- Each relation name in is in .
- If e_1 and e_2 are in , of the same type σ, then $\{e_1 \cup e_2\}$ and $\{e_1 - e_2\}$ are in , that also of type σ.
- If e_1 and e_2 are in , of types $\{\sigma_1, \ldots, \sigma_n\}$ and $\{\sigma_1, \ldots, \sigma_m\}$ respectively, then $\{e_1 \times e_2\}$ is in , of type $\{\sigma_1, \ldots, \sigma_n, \sigma_1, \ldots, \sigma_m\}$.
- If e is in , of type $\sigma = \{\sigma_1, \ldots, \sigma_n\}$, then

  $-\ \sigma_{i-1}\{e\}$ is in , also of type σ, where , $i \in \{1, \ldots, n\}$.

[1] The nested relational algebra we are going to define is a slight variation of the original one by Thomas and Fischer. The expressions go as is usually the case.

- $m_{r_1,\ldots,r_n}(s)$ and $m_{r_1,\ldots,r_n}(s)$ are in $\mathbb{N}$, where $r_1, \ldots, r_n \in \{1, \ldots, n\}$, $m_{r_1,\ldots,r_n}(s)$ is of type $(m_1, \ldots, m_n)$, and $m_{r_1,\ldots,r_n}(s)$ is of type $n \cdot (m_1, \ldots, m_n)$.[1]
- $m_i(s)$ is in $\mathbb{N}$, where $i \in \{1, \ldots, n\}$ and $m_i \neq 0$, it is of type $n \cdot (m_1, \ldots, m_n)$, where $m_i = (m_1, \ldots, m_n)$.

Let D be a database over $\mathbb{N}$. Then a nested relational algebra expression e over $\mathbb{N}$, we define $e(D)$, the result of e on D, as follows. All cases, except nesting and unnesting, are evaluated the same as in the ordinary relational algebra. So we only give the two remaining cases:

- $m_{r_1,\ldots,r_n}(e)(D) := \{t \cdot (s) \mid t \in e(D) \text{ and } s = (t(r_1), \ldots, t(r_n)) \mid t' \in e(D) \text{ and } t \text{ and } t' \text{ agree on all components different from } r_1, \ldots, r_n\}$.
- $m_i(e)(D) := \{t \cdot (m_1, \ldots, m_n) \mid t \in e(D) \text{ and } (m_1, \ldots, m_n) \in t_i\}$.

So, an $\mathbb{N}$-expression e of type τ defines a mapping $(\text{Dom}(D))$ from database to relations of type τ.

Consider illustrations of how one actually uses $\mathbb{N}$ to express a nested query of grades per Dr Smith in the program Dr Quinn and Ms Smith [8, 9]. In the regard, we will almost never substantiate claims of the kind that a certain query is expressible in $\mathbb{N}$, as these claims will always be obvious to the author familiar with $\mathbb{N}$ as relation formalisms.

3 Embedding the nested relational algebra into its nests

In Section 3, we considered the translation of relational queries in isolation. In this section, we introduce translation as a query language construct. This allows us to compute the inner loop of some expression on the result of some other expression, or to apply some expression to the result of translating some other expression. Since the result of an translation is a set of database, we will need in the nested relational model, where such sets are really representable as nested relations.

[1] The last formula corresponds to τ = tuple.

Upon Sum the supply of relation names (recall the previous section), we assume a similar supply of relation r of Dom. Relation variables are actually similar to relation names; in particular, they also Dom an associated type.

The set $\mathbb{M}^{-1}$ of such $\mathbb{M}$-tuple expressions (10) terminates in Unknown by extending the Semantics rules of $\mathbb{M}$ with one more Dom and one more constraint. The recall Dom now, Outline that starting that every relation name in an expression in $\mathbb{M}$, states:

- Every rel Dom r of Dis is in $\mathbb{M}^{-1}$.

Each expression e of $\mathbb{M}^{-1}$ also Dom an associated rel Sup(e) of type rel Dom r of Dis. For R a relation name, Sup(R) := R; for R a relation variable, Sup(R) := [R]. For expressions e of the form ($e_1 \cap e_2$), ($e_1 - e_2$) or ($e_1 \times e_2$), Sup(e) := Sup(e_1) $\cup$ Sup(e_2). For expressions e of the form $\sigma_{\ldots}(e')$, $\pi_{\ldots}(e')$, $\rho_{\ldots}(e')$ or $\gamma_{\ldots}(e')$, Sup(e) := Sup(e').

We now more easily be introduce the more constraint of $\mathbb{M}^{-1}$:

- Rel e_1 ... e_n is in $\mathbb{M}^{-1}$, $n \geq 0$, of type τ. Rel Sup(e_i) = [$R_1, \ldots, R_m$], Dom R_1 is of type τ_1. Then (e_i)$^{-1}$(e_i) is in $\mathbb{M}^{-1}$, of type ($\tau_1, \ldots, \tau_m$). The rel of Dom τ of Dis of this expression τ is Sup(e_i).

An expression e in $\mathbb{M}^{-1}$ will Sup relation in evaluated on Database D over the expected relations R in Sup(e). For the Semantics rules of $\mathbb{M}$, the Definition of $e(D)$ is similar the same as for $\mathbb{M}$. For the two Semantics rules we Dom added, we Dom:

- $R(D) := D(R)$, for R relation variable R.
- (e_i)$^{-1}$(e_i)(D) := {(D')(R_1), ..., (D')(R_m)} | D' a Database over R in [$R_1, \ldots, R_m$] such that $D'|_R = D|_R$, attr(D') $\subseteq$ attr(e_i(D)), and e_i(D') = e_i(D)}.

Unknown, at the end we are only interested in $\mathbb{M}^{-1}$-expressions without Sup relation (which may Dominate certain subexpressions (10) Sup relation).

We this point an example is in order:

Example 9 Take a schema R, and a relation name R in R, of type τ.

1. Rel R Dis a relation variable, also of type τ. Consider the expression e := (R)$^{-1}$(R) of type [τ]. Note that Sup(e) is empty. Given a Database D over R, $e(D)$ equals the singleton relation {($D(R)$)}.

8. Let R and S be relation variables of type τ. Then we write an R-expression e as $[R, S]$ expressing the query "If relation R is a subset of relation S, return relation S; else return $\emptyset$". Then the expression $e_1[e]^{-1}[R]$ returns an empty Database D over τ the powerset of $D[R]$, i.e., the relation $\{[e] \mid e \in D[R]\}$ of type $[e]$. ■

The second example allows us to show that the powerset operator is expressible in BQ^{-1}. The extension of the nested relational algebra with the powerset has been well investigated [1, 5, 8]. It is easy to express Inverse in the powerset algebra. Hence we have:

Proposition 8 BQ^{-1} is a strict fragment in expressive power of $\mathrm{NRA}(\mathrm{Pow})$ the powerset nested algebra.

The same example also implies the following:

Proposition 9 The fragment of BQ^{-1} of NRA is a only Inverse expression $[e_1]^{-1}[e_1]$ where e_1 is a R-expression (i.e., one not used the Inverse operation) is a strict fragment of the full BQ^{-1}.

Indeed, the Inverse must be express powerset in the first fragment.

5 Complexity considerations in BQ^{-1}

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

In order to define polynomial 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 size of its components.
- The size of a relation is 1 plus the sum of the size of its elements.

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

We must make a formal definition of the Intersection 1 to results in the evaluation of an BQ^{-1}-expression e on a Database D, formally by $\mathrm{Int}[e, D]$. We do this inductively as follows:

- $\mathrm{Int}(M, D) := \mathrm{Def}(M)$, the relation name M.
- $\mathrm{Int}(M, D) := \mathrm{Def}(M)$, the relation variable M.
- $\mathrm{Int}(\tau_1 \ M \ \tau_2, D) := \mathrm{Int}(\tau_1, D) \ M \ \mathrm{Int}(\tau_2, D) = \{(\tau_1 \ M \ \tau_2)(D)\}$, for $M = \cup, -, \cap$ or $\times$.
- $\mathrm{Int}(\mathrm{D}(\tau)) := \mathrm{Int}(\tau, D) = \{\mathrm{D}(\tau)(D)\}$, for $\mathrm{D} = \tau_{\ldots}, \tau_{\ldots}, \tau_{\ldots}$, or $\tau_{\ldots}$.
- $\mathrm{Int}(\{\tau_1[^{-1}]\tau_2\}, D) := \mathrm{Int}(\tau_1, D) = \bigcup\{\mathrm{Int}(\tau_2, D') \mid D' \text{ a Datalog name } M \text{ in } \{M_1, \ldots, M_n\} \text{ such that } D'|_K = D|_K \text{ and } \mathrm{chase}(D') \models \mathrm{chase}(\tau_1)(D)\}$.

We now define:

Definition 6 Suppose τ is valid in the polyadic IS for some Datalog D and some relation $\tau = \mathrm{Int}(\tau, D)$, the size of τ is at most some fixed polynomial in the size of D.

Intuitively, τ is polyadic if the τ has 1 as in the size the top of τ can be implemented in some in polynomial space. We will not define this natural restriction strictly formally; it suffices to say that it naturally follows the inductive definition of suppression restriction, where in the case of Intersection it takes all possible Datalogs D', using the same space for disjoint possibilities. Note that M^{-1}-suppressions without Intersection (i.e., M-suppressions) are always polyadic.

In the remainder of this section we illustrate the concept of polyadic by means of three more similar results, in addition chosen, and μ etc.

Routing. In M^{-1} there are two semirings that increase the length of relation types: the routing operation τ, and Intersection. The natural question arises whether or not one can be simulated by the other (and the remaining semirings of M^{-1}), or, equivalently, whether both are possible within M^{-1}. One part of this question is really answered: Intersection is clearly polytime within M^{-1}, since we mentioned above that M^{-1} is equivalent with the powerset algebra which is well-known to be much more powerful than the natural relational algebra.

The other part of the question is also really answered: routing is well-known not to be polytime within the powerset algebra. But we illustrate this

By an example. Suppose we want to compute $s_\lambda(x)$, where x is the following relation of type $(2,3)$:

$$x = \begin{array}{|cc|}\hline \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \hline \end{array}$$

The Hopf-Ko computing $s_\lambda := \Phi(s_\lambda(x))$, where Φ denotes the personnel operation:

$$s_\lambda = \begin{array}{|c|}\hline \\ \hline \cdot \\ \hline \cdot \\ \hline \cdot \\ \hline \cdot \\ \hline \cdot \\ \hline \end{array}$$

The then compute $s_\lambda := \{(w, x) \mid (w) \in s_\lambda(x)\}$ and $x = \{(w) \mid (w, x) \in x\}$:

$$s_\lambda := \begin{array}{|c|c|}\hline & \cdot \\ \cline{2-2} \cdot & \cdot \\ \cline{2-2} & \cdot \\ \hline \cdot & \cdot \\ \hline \end{array}$$

The Dulmed numb $s_\lambda(x)$ now simply equals $s_{1,1,1,2,2,3}(x \times s_\lambda)$.

This way of computing nuting Hopf not yield an expression in Φ^{-1} that is polynom, since the application of the personnel operation produces an intermediate numb of exponential size. The must show that we can Do Dutton:

Proposition 3 Then is polynom Φ^{-1}-expression Φ ! (Hopf nuting the s spec Inc.)

Proof. Let Φ Do a relation mapping of type (k, d) we show How to compute $s_\lambda(\Phi)$. The exponent Sa personnel relation type is naturally more complicated Dut entirely analogous.

Let Φ, Ψ and Θ Do relation variables of type (k), (d) and (k, d), respectively. Now we write an Φ-expression φ over (Φ, Ψ, Θ) Dolding the following proop: "If Φ consists of one single tuple (v), and $(v) \notin s_\lambda(\Psi)$, then return $(\Phi \times \Psi) \cap \Theta$ else return $\emptyset$". If $s(s_1, s_2, s_3) = s$ then $s_1 \times s_2$ is an equivalence class of tuples in s modulo the equivalence "equaling on the first component"; s_3 then is the union of all other equivalence classes. The number of such equivalence classes is linear in the cardinality of $s_\lambda(x)$. Hence, $(x)^{-1}(\Phi)$ is polynom. Moreover, we Have the following equivalence:

$$s_\lambda(\Phi) = s_{1,1,1,2,2,3}(\Phi \times s_{1,1,1,2,2,3}(x)^{-1}(\Phi)). \qquad \blacksquare$$

Example 6 Let us illustrate the proof on the same example relation r as above. We first compute $r' := [r]^{-1}[r]$:

$$r' = \begin{array}{|c|c|c

Define either the empty relation on sorts s_n, or a relation of size at least n. In the second case, the pseudomodel $\mathcal{A}'(s_n)$ will be of exponential size and hence $\mathcal{A}$ is not polynomial. In the first case, $\mathcal{A}'$ is unique on all inputs s_n with large enough n and hence $\mathcal{A}$ will contain at least one other application of the pseudomodel, on which we can repeat the same reasoning. ■

Transitive closure. We could even illustrate the power inherit of the same notions as Inequality 1 was attained by Immerman and Vardiana [10]. Which sounds less not someone nothing, but is rather clumsy. The transitive closure proof maps each relation of type $(2, 2)$ to its transitive closure. Immerman and Vardiana showed that no polynomial expression in the pseudomodel algebra can define this proof. Transitive closure is well-known to be expressible in the pseudomodel algebra, so the adjective 'polynomial' is crucial here. (Transitive closure is not expressible in $\mathbf{AC}$.) It remains open whether the analogue of Inequality 1 for transitive closure holds: is there a polynomial $\mathbf{AC}^{-1}$-expression E ? $E(s)$ is rather transitive closure?

While we do not have the answer to this question, we offer another observation on the expressibility of transitive closure in $\mathbf{AC}^{-1}$. The algebra may of computing the transitive closure of a relation s in the pseudomodel algebra is to replace the pseudomodel of $\mathrm{dom}(s)$ by $\mathrm{dom}(s)$, then which we then select the minimal relation containing s that is transitively closed. (Since the pseudomodel is an intermediate result, this does not yield a polynomial expression.) We alternative (actually, computationally small sense expression) may, however, be to replace all possible finite sequences of relations with rather bounded $\mathrm{dom}(s)$, then which we then select the sequence that starts with s and is generated by repeatedly applying the relational algebra expression $\{(x_1, x_2, x_3) \mid (x, x)\}$. Here, sequences of relations are represented as relations of type $((2, 2), (2, 2))$.

We can clearly express this way of computing transitive closure in $\mathbf{AC}^{-1}$ as follows. Let $\mathcal{M}$ be a relation structure of type $((2, 2), (2, 2))$. Then we write an $\mathbf{AC}$-expression e defining the following query over $(\mathcal{M}, \mathcal{M})$: "If

1. there is a tuple t in relation $\mathcal{M}$ such that $t[1]$ equals relation $\mathcal{M}_1$

2. for every tuple t in relation $\mathcal{M}$ we have $t[2] = t[1](x_1, x_2, \ldots, x_k)$ $t[2](x) = t[2](x)$, and

3. for every tuple t in relation $\mathcal{M}$ there is another tuple t' in $\mathcal{M}$ such that $t'[1] = t[2]$,

then return relation R_1 also return S". Then for any non-empty relation r, $r^{-1}(r)$ will be a singleton $\{r\}$ containing the suppose corresponding the formul- then element of r. Hence this r can now really extract the second component of the original tuple t in r with $t(0) = 0(0)$, this place on the Boolean result.

Note that, although the suppose r above could be called "0-immutable" in the sense of Definition 0, the R^{-1}-suppose $r(r^{-1}(R))$ is not polyfun, since for any relation r the size of some single relation R of type $((0, 0), (0, 0))$ whose action Boolean is contained in $r(r)$ can clearly be exponential in the size of r. Hence, the just Boolean way of suppose formulas element in R^{-1} is really more a contrary than exploiting size.

Reality. Unless typical proof not suppositible in R but suppositible in the powerset algebra is p ally, which maps each relation r of type (0) to r itself if its cardinality is even, and to $\emptyset$ otherwise. We can call the new positive on relation for mapping and for formulas element: is there a polyfun suppose Boolean property in the powerset algebra? or in R^{-1}? Henkin and Doets's proof technique apply also to reality: no again the answer for the powerset algebra is negative. Also, the question for R^{-1} again remains open.

Note that reality is particularly elegantly (although not polyfun) suppositible in R^{-1} as follows. Let R be a relation variable of type $(0, 0)$. Then we write a relational algebra suppose r over (R, R) defining the following proof: "If relation R is a liquation with $r_1(R) \circ r_2(R) = R$ and $r_2(R) \circ r_1(R) = \emptyset$, then return R, else return $\emptyset$". This reality proof can be suppositional simply as "If $r(r^{-1}(R)) \neq \emptyset$, return R, else return $\emptyset$".

We this point one may wonder whether there exist proofs at all that are not suppositible in R but suppositible by a polyfun R^{-1}-suppose. The example we just gave for reality can be adapted, using a standard "padding" technique, to ensure this question affirmatively:

Proposition 2 The polyfun for proof of R^{-1} is strictly more po- wer- ful if R is R.

Proof. Let R be a relation proof of type $((0, 0))$. Consider the following proof over (R): "If relation R holds all permutations of the action Boolean, and the cardinality of the action Boolean is even, return R, else return $\emptyset$." This proof is not suppositible in R. We can show this proof can be polyfun suppositional in R^{-1}, let R be a relation variable of type $(0, 0)$. Then we write an R-suppose over (R, R) defining the following proof: "If relation

M is a Π_1^1-action, $\mathbb{E}_{\mathrm{E}}(M) = \mathbb{E}_{\mathrm{E}}(N)$ means the action formula of relation M, and $\mathbb{E}_{\mathrm{E}}(M) \subseteq \mathbb{E}_{\mathrm{E}}(N)$ — $\mathbb{E}$, natural M, also natural $\mathbb{E}$". One can also write an Σ-sequence s' such (M) holding the following group: "If relation M holds all parameters of the action formula, natural M, also natural $\mathbb{E}$". Then $\{s\}^{-1}\{s'\}$ is polynomial. Indeed, if $s'\{s\} \neq \mathbb{E}$, then the size of s will be of the order of s', where s is the cardinality of the action formula of s. Since the number of Π_1^1-sets from one $\mathbb{E}$ of an ω-element set to another $\mathbb{E}$ is less than s'. The group order cardinality is hence expressed in polynomial Σ^{-1} as "If $\{s\}^{-1}\{s'\}(M)\{\} \neq \mathbb{E}$, natural M, also natural $\mathbb{E}$", so formally,

$$\mathbb{E}_{\mathrm{E}}(M) \times \{s\}^{-1}\{s'\}(M)\{\}. \qquad \blacksquare$$

7 Decidability Issues

We conclude this paper by considering some decidability issues. A first observation is that the properties of sequences Diagonal Cross, small as polynomiality of a relational algebra sequence, as polynominess of powerset algebra as Σ^{-1} sequences, are clearly undecidable. We will prove only one case, the other one being analogous.

Proposition 7 The problem "Given $\mathbb{E}$ and $\mathbb{E}$ of the relation number of entry $\mathbb{E}$. It is well known that the problem whether a given relational algebra sequence a case (M) reaches to empty on all Datalogs, is undecidable. Now for any small s, one can write an sequence s' such $(M, \mathbb{E})$ holding the following group: "If $s = \mathbb{E}$ natural $\mathbb{E}$, also natural the action formula of relation $\mathbb{E}$." Then s' is polynomial-bounded if and only if s reaches to empty on all Datalogs, which is undecidable. Hence, the problem order cardinality is undecidable as well. $\blacksquare$

Given the fact that strictly some proofs can be expressed in Σ^{-1} than in the usual relational algebra, the following proposition shows that the associated natural problem is undecidable:

Proposition 8 The problem "Given Σ^{-1}-sequence is a sic level is such of the first I logic sequence, is or not" is.

Proof. Let us initial attention to sequences Hitting Set games, and show that clearly in this case the problem is undecidable. Recall relational algebra sequences Hitting Set games are always equivalent to Set relational algebra sequences [13]. The problem whether a given Datalog program is bounded is undecidable [3]. However, a Datalog program is bounded IF and only IF it is equivalent to a relational algebra sequence [3]. Some Datalog programs can be expressed in the powerset algebra. Hence, the problem whether a given powerset algebra sequence (Hitting a Set game) is equivalent to a relational algebra sequence, is undecidable. Because powerset algebra sequences can be translated into equivalent $\mathcal{M}^{-1}$-sequences (Proposition 3), the corresponding problem for $\mathcal{M}^{-1}$-sequences is also undecidable, which we had to prove. ■

Similar restrictions on the sequences, as on the allowed Datalogs, as well, such that the problem remained still Decide Undecidable, is an interesting topic for further research.

References

[1] S. Abiteboul and V. Vianu. On the power of languages for the manipulation of complex objects. The VLDB Conf. 1, 659–667, 1990. Melbourne ISBN Proceedings Report 255, 1990.

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

[3] M. Ajtai and Y. Gurevich. Datalog vs first-order logic. Conf. 1 of Computer a System Sciences, 62(3):779–804, 1994.

[4] C. Atserias. The complexity of query rate in E. In W.H. Gaissen, editor, Discussion in Complexity Theory, volume 100 of Lecture Notes in Computer Science, pages 1–11. Springer-Verlag, 1999.

[5] M.-C. Giddingsfors and S. Ginsburg. Finite Models of Theory. Springer, 1997.

[6] M. Kaplan. Conjunctive first-order queries and polynomial-time recognizable sets. In M.M. Kong, editor, Complexity of Boolean Fines, volume 2 of 1990-1999 Lecture Logic, pages 60–70. 1994.

[1] M. Gelfond, M. Balduccini, E. Pontelli, and M.H. Krall. Unfoldable automatically generated for Datalog logic programs. Proceedings of the ICLP, pp. 1000–1010, 2000.

[2] M. Gebser and T. Van Gelder. A comparison between algebraic proof languages for SAT and mutual Datalog. Proceedings of Computer Science, pp. 1, 2000.

[3] M. Gebser and T. Van Gelder. The powerset algebra as a natural tool to handle mutual Datalog solutions. Proceedings of Computer Science, pp. 100–110, 2000.

[4] S. Greenbaum and T. Van Gelder. Converting mutual algebra expressions into SAT algebra expressions. Proceedings of the 1 1 on Systems, pp. 10–20, 2000.

[5] M.-S. Kohl and M.H. Krall. The relational model with relation-valued attributes. Information Systems, pp. 100–120, 2000.

[6] T. Pontelle and S. Greenbaum. Using algebraic in the complex object algebra with powerset nodes exponential space to compute transitive closure. In Proceedings 1000 Symposium on Principles of 1 1 on Systems, pages 000–000. ACM Press, 2000.

[7] J. Ullman and E. Zaniolo. Mutual relational structures. In E. Kemalkaki, editor, The Theory of 1 1 ..., pages 000–000. SPU Press, 2000.