Bounded Fixpoints for Complex Objects

Dan Suciu
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, USA

Abstract

We investigate a query language for complex-object databases, which is designed to (1) express only tractable queries, and (2) be as expressive over flat relations as first order logic with fixpoints. The language is obtained by extending the *nested relational algebra* $\mathcal{NRA}$, of [9], with a "bounded fixpoint" operator. Extending the flat case [15, 21] , all $PTime$ computable queries over *ordered* databases are expressible in this language. The main result consists in proving that this language is a conservative extension of the first order logic with fixpoints, or of the *while*-queries (depending on the interpretation of the bounded fixpoint: inflationary or partial). The proof technique uses indexes, to encode complex objects into flat relations, and is strong enough to allow for the encoding of $\mathcal{NRA}$ with unbounded fixpoints into flat relations. We also define a complex object logic based language with fixpoints and prove that its range restricted fragment is equivalent to $\mathcal{NRA}$ with bounded fixpoints.

1 Introduction

Several query languages for databases with complex objects have been studied in recent years (e.g. [1], [2], [3], [9], [12], [13], [14], [17], [20], [22]). A natural way of designing such a language is to extend first order logic to a logic for hereditary finite sets, and consider only *domain independent* queries, like in the case of first order logic. Abiteboul and Beeri follow this path in [1], define *safe* queries, and show that the resulting language, which they call *the calculus*, can express *powerset*, hence, unlike first order logic, it can express untractable queries. They design an algebraic language equivalent to *the calculus*, called *the algebra*, in which *powerset* is one of the primitives. Searching for a tractable sublanguage, they observe that by removing the powerset from the algebra one gets such a language, and that the same expressive power can be obtained by enforcing stricter safety rules in the calculus (the resulting language is called *the strictly safe calculus*).

Another way, based on an investigation of the primitives associated with the types occurring in complex objects, has been pursued by Breazu-Tannen Buneman and Wong ([9]), who designed a tractable language with the same expressive power as the algebra without powerset. Following established tradition, they called this language the *nested relational algebra* ($\mathcal{NRA}$), and showed that, like in first order logic, all queries expressible in $\mathcal{NRA}$ are in PTIME. We shall adopt their formalism in this paper.

Paredaens and Van Gucht in [17] and Wong in [23] show that $\mathcal{NRA}$, or languages equivalent to it, are conservative extensions of first order logic, meaning that using intermediate results of heigher set complexity does not help us expressing more queries. For example, the transitive closure of a flat relation cannot be computed in these languages, because it cannot be computed in first order logic. In contrast, in [1] it is shown that one can express transitive closure in the *algebra* (which has the same expressive power as $\mathcal{NRA} + powerset$), but in an inefficient way, by using more complex intermediate types. In fact, Hull and Su prove in [13] that the expressive power of this language becomes strictly more powerful as we allow intermediate types of larger set heights.

Transitive closure can be expressed in first order logic *with (inflationary) fixpoints*, a language equivalent to $DATALOG^{\neg}$ (see [6] for a review). Adding fixpoints to first order logic keeps the queries tractable, but adding them to $\mathcal{NRA}$ does not: we show that they are equivalent to *powerset* (see example 2 and proposition 3). It is natural to search for an extension of $\mathcal{NRA}$ which is still tractable and which has at least the expressive power of $DATALOG^{\neg}$. Peter Buneman suggested adding a *bounded fixpoint* and conjectured the conservativity result (theorem 2). When $f : \mathcal{P}(D) \rightarrow \mathcal{P}(D)$ is a function, and A is a subset of D, then the *bounded* fixpoint of f is the fixpoint of the function $g(x) \stackrel{\text{def}}{=} f(x) \cap A$. We consider both *inflationary* and *partial* semantics for the (bounded) fixpoints (see e.g. [4], or section 3 of this paper), and we denote them with $bfix_i$ and $bfix_p$. We prove in proposition 5 (section 3) that all queries in $\mathcal{NRA} + bfix_i$ are in $PTime$, and that all queries in $\mathcal{NRA} + bfix_p$ are in $PSpace$.

A further contrast between the fixpoint over flat types and the fixpoint over complex objects is given by proposition 2, which shows that both inflationary and partial fixpoint have the same expressive power over complex objects: Abiteboul and Vianu prove in [7], that the inflationary and partial fixpoint have the same expressive power over first order logic iff $PTime = PSpace$, which most people believe to be false.

We also present a logic based language having the same expressive power as $\mathcal{NRA} + bfix$. Grumbach and Vianu in [12] also present a tractable logic based language which can express recursive queries, but we do not know its precise expressive power, or whether it is closed under *map* (as our language is). We hope to be able to clarify in the future the relationship between our language and that presented in [12]. We remark that the range restriction rules in [12] differ substantially from ours.

The main result of this paper is the conservativity property of $\mathcal{NRA} + bfix$ over first order logic with fixpoints. We emphasize that this result holds for *arbitrary* databases: in fact, in the presence of order, this would be an immediate consequence of Immerman and Vardi's results, that first order logic with fixpoints can express all $PTime$ ($PSpace$) queries over ordered databases ([15], [21]). For the proof, we use a different technique than the ones used in [17] and [23], because it is not clear how to adapt their reduction methods for the bounded fixpoint. It also differs from the proof of conservativity results in [16], because we do not have order. Our approach is of a semantic nature, and splits in two steps the translation of $\mathcal{NRA} + bfix$ into the relational algebra with fixpoints: first, we encode all complex object into flat relations, using indexes from some infinite set I. In fact, this encoding is strong enough to translate even the unbounded fixpoint. We have learned that indexes have been used in a related way by Van den Bussche ([20]) for proving a conservativity result for the nested algebra (without fixpoints). In the second step, we show how to eliminate the indexes when both the input and the output of some relational algebra expression with indexes, are encodings of flat relations. It should be emphasized that, although I has some additional structure, no order relation or additional structure is needed on the other domains. Finally we observe that the bounded fixpoint over flat relations has the same expressive power as the unbounded fixpoint. In fact, by composing the transformations performed in these two steps, we do get a direct translation of $\mathcal{NRA}+bfix$ into the relational algebra (without indexes).

In fact, our result is slightly more general, because it holds also when the language contains *interpreted* function symbols, over base types and flat relations. In that case, $\mathcal{NRA} + bfix$ is a conservative extension of the relational algebra with *bounded* fixpoints, because only in the absence of interpreted function symbols has the bounded fixpoint over flat relations the same expressive power as the unbounded fixpoint.

An immediate consequence of the conservativity result is the fact that, in general, there are queries in $PTime$ which are not expressible in $\mathcal{NRA} + bfix$ (e.g. the test whether a flat relation has an even number of elements). But, as in the flat case (see [21] and [15]), we can prove that any $PTime$ or $PSpace$ database transformation over *ordered* databases is expressible in $\mathcal{NRA} + bfix$.

In [16], Libkin and Wong prove for a query language with arithmetic, order, and bounded fixpoints, that, for any $k \geq 1$, it is a conservative extension of its restriction to types of set height $\leq k$. This result, and our conservativity result, do not imply each other. Our result does not *require* the presence of order and arithmetic (but still holds in the presence of additional interpreted function symbols), however we consider only the case $k = 1$.

In [18] Saraiya shows that $DATALOG$ can be simulated with structural recursion on sets, preserving the PTIME complexity, by using as an intermediate step a loop operator which iterates a function a number of times equal to the cardinality of a set. Several complexity-theoretic results for program properties and transformations are then be obtained by recourse to known results for $DATALOG$.

In section 2, we give the basic definitions of $\mathcal{NRA}$. In section 3, we define the unbounded and the bounded fixpoint, state some basic properties, and give a simple proof of the equivalence between the expressive power of the inflationary and partial fixpoint, over complex objects. Section 4 states our main results. Section 5 presents a logic based language called $\mathcal{NRCALC} + fix$, which is equivalent to $\mathcal{NRA} + bfix$. In section 6 we describe the encoding of the nested relational algebra into the relational algebra with indexes, then, we prove in section 7 that indexes can be eliminated, for flat relations. Together, these prove the conservativity extension theorem.

2 The Nested Relational Algebra

We consider a set of base types, like *nat*, *string*, etc., to be given, and define **types** by the grammar $\sigma = b \mid unit \mid \sigma_1 \times \sigma_2 \mid \{\sigma\}$ (b stands for a base type, $unit = \{()\}$, $\sigma_1 \times \sigma_2$ is the product type, and $\{\sigma\}$ contains all finite subsets of σ). For the particular set of base types, we also assume a given collection of external functions to be given, each function p with a specified domain d_p and codomain c_p, written $p : d_p \to c_p$. Examples are *cardinality* at type σ ($card_\sigma : \{\sigma\} \to nat$), *addition* ($add : nat \times nat \to nat$), the constant *zero* ($0 : unit \to nat$) etc. Note that function symbols of arity $unit \to b$ play the role of a constant of type b. Let Σ be the set of base types and external functions. Sometimes we restrict Σ to having only base types and constants (i.e. function symbols of type $unit \to \sigma$): we shall write $\Sigma = \emptyset$ in this case, although Σ is *not* necessarily empty (it may contain base types and constants). Following [9], we define the **nested relational algebra** $\mathcal{NRA}(\Sigma)$ as an algebra of functions over complex objects, without variables. The functions in this language are defined by the following rules:

$$\frac{p \in \Sigma}{p : d_p \to c_p} \qquad \frac{}{id_\sigma : \sigma \to \sigma} \qquad \frac{f : \sigma \to \tau \quad g : \tau \to \nu}{g \circ f : \sigma \to \nu}$$

$$\frac{}{\pi_i : \sigma_1 \times \sigma_2 \to \sigma_i}(i = 1, 2) \qquad \frac{f_i : \sigma \to \tau_i \ (i = 1, 2)}{(f_1, f_2) : \sigma \to \tau_1 \times \tau_2}$$

$$\frac{}{\iota_\sigma : \sigma \to unit}$$

$$\frac{}{\eta_\sigma : \sigma \to \{\sigma\}} \qquad \frac{}{\mu_\sigma : \{\{\sigma\}\} \to \{\sigma\}} \qquad \frac{f : \sigma \to \tau}{map(f) : \{\sigma\} \to \{\tau\}}$$

$$\frac{}{\rho_{2,(\sigma,\tau)} : \sigma \times \{\tau\} \to \{\sigma \times \tau\}}$$

$$\frac{}{\emptyset_\sigma : unit \to \{\sigma\}} \qquad \frac{}{\cup_\sigma : \{\sigma\} \times \{\sigma\} \to \{\sigma\}}$$

$$\frac{}{not : \{unit\} \to \{unit\}} \qquad \frac{}{eq_b : b \times b \to \{unit\}}(b \text{ a base type})$$

We abbreviate $\mathcal{NRA}(\Sigma)$ by $\mathcal{NRA}$ when $\Sigma = \emptyset$ (i.e. Σ contains only base types and constants). The semantics of these operations is fully described in [9]. Here, we briefly sketch it: π_1, π_2 are the projections, $\iota_\sigma(x) \stackrel{\text{def}}{=} ()$, $\eta_\sigma(x) \stackrel{\text{def}}{=} \{x\}$, $\mu_\sigma(\{x_1, \ldots, x_n\}) \stackrel{\text{def}}{=} x_1 \cup \ldots, \cup x_n$, $map(f)(\{x_1, \ldots, x_n\}) \stackrel{\text{def}}{=} \{f(x_1), \ldots, f(x_n)\}$, $\rho_{2,(\sigma,\tau)}(x, \{y_1, \ldots, y_n\}) \stackrel{\text{def}}{=} \{(x, y_1), \ldots, (x, y_n)\}$, $\emptyset_\sigma$ returns the empty set, $\cup_\sigma$

is set union, $not(\emptyset) = \{()\}$ and $not(\{()\}) = \emptyset$, $eq_b(x,x) = \{()\}$ and $eq_b(x,y) = \emptyset$ when $x \neq y$.

Following [9], we use the type $\{unit\}$ as booleans, using the convention that $\emptyset$ stands for $false$, and $\{()\}$ stands for $true$. Then, one can show (see also [9]) that in $\mathcal{NRA}$ we can express equality at all types $eq_\sigma : \sigma \times \sigma \to \{unit\}$ (*not* is necessary for this), the membership test $member : \sigma \times \{\sigma\} \to \{unit\}$, the *cartesian product* $\bowtie_{\sigma,\tau}$: $\{\sigma\} \times \{\tau\} \to \{\sigma \times \tau\}$[1] ($x \bowtie y \stackrel{\text{def}}{=} \{(a,b) \mid a \in x, b \in y\}$), set difference $- : \{\sigma\} \times \{\sigma\} \to \{\sigma\}$, intersection $\cap_\sigma : \{\sigma\} \times \{\sigma\} \to \{\sigma\}$, the conditional $if\ then\ else\ : \{unit\} \times \{\sigma\} \times \{\sigma\} \to \{\sigma\}$, $nest : \{\sigma \times \tau\} \to \{\sigma \times \{\tau\}\}$, $unnest : \{\sigma \times \{\tau\}\} \to \{\sigma \times \tau\}$, the database projections $\Pi_i : \{\sigma_1 \times \sigma_2\} \to \{\sigma_i\}$ ($\Pi_i \stackrel{\text{def}}{=} map(\pi_i)$, $i = 1, 2$), etc. Also, for $f : \sigma \to \{\tau\}$, one defines $ext(f) : \{\sigma\} \to \{\tau\}$ to be $\mu_\tau \circ map(f)$ (see [9]): the meaning of ext is $ext(f)(\{x_1, \ldots, x_n\}) = f(x_1) \cup \ldots, \cup f(x_n)$. We shall eliminate type indexes, when they are understood from the context, i.e. we shall write μ instead of μ_τ, etc.

In [9] it is shown that all functions expressible in $\mathcal{NRA}(\Sigma)$ are in $PTime$, provided that all functions in Σ are in $PTime$. Moreover, they show that $\mathcal{NRA}$ is equivalent to the *strictly safe calculus* or *the algebra without powerset* in [1]. It is also (almost) equivalent to the *nested algebra* in [17] (see [16]).

Following [13], we define the **set height of a type** σ to be: $sh(b) = sh(unit) = 0$, $sh(\sigma \times \tau) = max(sh(\sigma), sh(\tau))$, $sh(\{\sigma\}) = 1 + sh(\sigma)$. The **set height of a function** $f : \sigma \to \tau$ is $sh(f) \stackrel{\text{def}}{=} max(sh(\sigma), sh(\tau))$.

For $k \geq 0$, we want to define $\mathcal{NRA}_k(\Sigma)$ to be the sublanguage of $\mathcal{NRA}(\Sigma)$ whose types have set height $\leq k$. We encounter, however, the following problem: consider some function $f : \sigma \to \{\tau\}$ in $\mathcal{NRA}_k(\Sigma)$, for which $sh(\sigma) = k-1$, and $sh(\{\tau\}) \leq k$. We would like $ext(f) : \{\sigma\} \to \{\tau\}$ to be also in $\mathcal{NRA}_k(\Sigma)$, but the latter is defined as $\mu \circ map(f)$, and uses an intermeditate type of set height $k+1$, hence it cannot be expressed in $\mathcal{NRA}_k(\Sigma)$. So, we define $\mathcal{NRA}_k(\Sigma)$ similar to $\mathcal{NRA}(\Sigma)$, with the following change:

- We remove map, μ and ρ_2, and add ext_2, a generalized version of ext:

$$\frac{f : \nu \times \sigma \to \{\tau\}}{ext_2(f) : \nu \times \{\sigma\} \to \{\tau\}}$$

  (with the semantics $ext_2(f)(x, \{y_1, \ldots, y_n\}) \stackrel{\text{def}}{=} f(x, y_1) \cup \ldots \cup f(x, y_n)$). Now $ext(f)$ is expressible as $ext(f) \stackrel{\text{def}}{=} ext_2(f \circ \pi_2) \circ (\iota_{\{\sigma\}}, id_{\{\sigma\}})$, $map(f)$ is expressible as $map(f) \stackrel{\text{def}}{=} ext(\eta \circ f)$, μ is expressible as $ext(id)$, and $\rho_2 \stackrel{\text{def}}{=} ext_2(\eta)$.

Fact 1 $\mathcal{NRA}(\Sigma) = \bigcup_{k \geq 0}(\mathcal{NRA}_k(\Sigma))$

[1] We use $\bowtie_{\sigma,\tau}$ instead of $\times_{\sigma,\tau}$ to avoid confusion with the bifunctor $\times$, e.g. in expressions like $\times_{\sigma,\tau} \times \times_{\sigma',\tau'}$ (meaning $\bowtie_{\sigma,\tau} \times \bowtie_{\sigma',\tau'}$).

Proposition 1 (Combinatorial Completeness) *Let $\mathcal{X}$ be some set of variables, i.e. function symbols of type $x : unit \to t_x$, where t_x is the type of x. Then $\mathcal{NRA}_k(\Sigma \cup \mathcal{X})$ is* **combinatorial complete**, *i.e. for any expression $f : \sigma \to \tau$ and any variable $x : unit \to t$, there is some expression $Kx.f : t \times \sigma \to \tau$ which doesn't contain the variable x[2], such that $f = (Kx.f) \circ (x \circ \iota_\sigma, id_\sigma)$.*

Proof (Sketch) The proof is done by induction on the size of the expression f. All base cases are trivial. When f is of the form $g \circ f$ or (f_1, f_2), then the proof is identical to [9]. It remains to consider the case when $f = ext_2(g)$, where $g : \nu \times \sigma \to \{\tau\}$ and $ext_2(g) : \nu \times \{\sigma\} \to \{\tau\}$. By induction, there is some $Kx.g : t \times (\nu \times \sigma) \to \{\tau\}$. Let $h : (t \times \nu) \times \sigma \to \{\tau\}$ be $Kx.g \circ (\pi_1 \circ \pi_1, (\pi_2 \circ \pi_1, \pi_2))$; then $Kx.ext_2(g) \stackrel{\text{def}}{=} ext_2(h) \circ ((\pi_1, \pi_1 \circ \pi_2), \pi_2 \circ \pi_2)$. To prove the equation for $Kx.ext_2(g)$ from the one corresponding to $Kx.g$, one uses $ext_2(f \circ (k \circ \pi_1, \pi_2)) = ext_2(f) \circ (k \circ \pi_1, \pi_2)$, for $f : \nu' \times \sigma \to \{\tau\}$ and $k : \nu \to \nu'$. $\square$

As a consequence, $\mathcal{NRA}_k(\Sigma)$ can be presented in a friendlier calculus-style language, with variables and lambda-abstractions (see [9] for details), and we shall feel free to use such a language in the sequel. For example, consider some predicate $p : \tau \to \{unit\}$ in $\mathcal{NRA}(\Sigma)$. We want to define the *selection* associated to p, $\sigma_p : \{\tau\} \to \{\tau\}$, like this: $\sigma_p \stackrel{\text{def}}{=} \lambda x.\Pi_2(ext(\lambda v.p(v) \bowtie \{v\})(x))$. We can translate this into $\mathcal{NRA}(\Sigma)$, by first introducing variables $v : unit \to \tau$ and $x : unit \to \{\tau\}$. Next, we translate expressions of some type ν into functions of type $unit \to \nu$, and we translate functions of type $\nu \to \nu'$, into functions of the same type. The definition of σ_p will be translated into: $(Kx.\Pi_2 \circ ext((Kv. \bowtie \circ (p \circ v, \eta \circ v)) \circ (id_\tau, \iota_\tau)) \circ x) \circ (id_{\{\tau\}}, \iota_{\{\tau\}})$.

As shown in [8], $\mathcal{NRA}_1$ is a conservative extension of the relational algebra (because one can see that $\mathcal{NRA}_1$ is equivalent to the language of $\mathcal{E}$'-expressions with free tuple variables and the emptiness test, as defined in [8]).

A natural question arises: can we express more functions using intermediate types of higher set height ? For example, it is known that we cannot express the transitive closure $tc : \{b \times b\} \to \{b \times b\}$ in $\mathcal{NRA}_1$ (see [19]): can we express it in $\mathcal{NRA}$, by using intermediate results of higher set heights, in some clever way ? The answer is negative:

Theorem 1 *([23], [17]) $\forall k \geq 1$, if all function symbols in Σ have set height $\leq k$, then $\mathcal{NRA}(\Sigma)$ is a conservative extension of $\mathcal{NRA}_k(\Sigma)$*

The condition imposed on the functions in Σ is obviously necessary. Indeed, suppose that Σ contains the powerset over binary relations $powerset : \{b \times b\} \to \{\{b \times b\}\}$ (a function of set height 2). Then, using the technique from [1], one can express transitive closure $tc : \{b \times b\} \to \{b \times b\}$, which cannot be expressed in $\mathcal{NRA}_1(\Sigma)$, where $powerset$ cannot be invoked.

As a consequence of this theorem, $\mathcal{NRA}$ is a conservative extension of the relational algebra.

[2] $Kx.f$ contains all variables in f, except x. The notation is analog to the $\lambda x.f$ syntax.

3 Fixpoints

First order logic and the relational algebra have been extended to strictly more expressive languages, like first order logic with inflationary fixpoints (which is equivalent to $DATALOG^{\neg}$ with inflationary semantics), or first order logic with partial fixpoints (which is equivalent to the *while*-queries; see [6] for a review). These two languages can express only queries in $PTime$ ($DATALOG^{\neg}$) or in $PSpace$ (*while*-queries), and they are in some restricted sense complete for the respective complexity classes, namely they can express *all* $PTime$ or $PSpace$ queries over *ordered* databases[3] ([15], [21]).

As a consequence of the conservativity result (theorem 1), $\mathcal{NRA}$ is, over flat relations, only as expressive as first order logic. So we investigate possible extensions, to make it as powerful as first order logic with fixpoints (inflationary or partial). But, by simply adding the obvious generalization of the fixpoints to higher order sets, we get a language whose queries are no longer in $PTime$ (or $PSpace$) ! We investigate an alternative fixpoint, the *bounded fixpoint* (an idea due to Peter Buneman), with inflationary or partial semantics. The resulting two languages turn out to be conservative extensions of first order logic with inflationary or partial fixpoints, express only queries in $PTime$ and $PSpace$ respectively, and express *all* queries in these complexity classes, over ordered databases.

Fixpoints cannot be defined at all types, but only at set types and products of set types. So we define a **product of sets type** (**PS type**) to be either a set type $\{t\}$, or the product $\sigma \times \tau$ of two PS types σ and τ. Examples of PS types are $\{b\}$, $\{b \times \{b\}\} \times \{b\} \times \{b \times b\}$ etc., while b and $b \times \{b\}$ are not PS types. For some PS type τ, we write $\cup, \cap : \tau \times \tau \to \tau$ and $\emptyset : unit \to \tau$, for the union and intersection of the components, and for the tuple of empty sets.

Consider the following fixpoint constructions, in which τ is a PS type[4]:

$$\frac{f : \sigma \times \tau \to \tau}{fix\ f : \sigma \to \tau} \qquad \frac{f : \sigma \times \tau \to \tau \quad g : \sigma \to \tau}{bfix(f \mid g) : \sigma \to \tau}$$

We consider two semantics: the *partial*, and the *inflationary* semantics (see [4]). To distinguish them, we write $fix_p\ f$ and $fix_i\ f$ (respectively $bfix_p(f \mid g)$ and $bfix_i(f \mid g)$). For $x \in \sigma$, define the sequence $y_0 = \emptyset\ (\in \tau)$, $y_{n+1} = f(x, y_n)$; if there is some n for which $y_n = y_{n+1}$, then we define the **partial fixpoint** of f to be $(fix_p\ f)(x) = y_n$. Else, $(fix_p\ f)(x) =$ undefined. The **inflationary fixpoint** is defined to be the partial fixpoint of $f_i(x, y) = y \cup f(x, y)$, or, equivalently, $(fix_i f)(x) = \bigcup_{n \geq 0} y_n$, where $y_0 = \emptyset$, $y_{n+1} = y_n \cup f(x, y_n)$. The **bounded** partial (inflationary) fixpoint of f and g, $bfix(f \mid g)$, is defined to be the partial (inflationary) fixpoint of $f(x, y) \cap g(x)$.

All four resulting languages $\mathcal{NRA}(\Sigma) + fix_p$, $\mathcal{NRA}(\Sigma) + bfix_p$, $\mathcal{NRA}(\Sigma) + fix_i$, $\mathcal{NRA}(\Sigma) + bfix_i$ are combinatorial complete. The first three can express *partial* (as opposed to *total*) queries. When $\Sigma = \emptyset$, then only the first two can express partial queries.

[3] No query language capable of expressing *exactly* the $PTime$ queries is currently known.

[4] The presence of the type σ in both definitions is necessary for combinatorial completeness.

Example 1 *The transitive closure can be computed using the bounded fixpoint: let* $f : \{\sigma \times \sigma\} \times \{\sigma \times \sigma\} \to \{\sigma \times \sigma\}$ *and* $g : \{\sigma \times \sigma\} \to \{\sigma \times \sigma\}$ *be:* $f(x, y) = x \cup (x \circ y)$ *(where* $x \circ y$ *is* relation composition*, which can be easily defined in* $\mathcal{NRA}$*),* $g(x) = (\Pi_1(x) \cup \Pi_2(x)) \bowtie (\Pi_1(x) \cup \Pi_2(x))$*. Then the transitive closure of* x *is given by* $tc(x) = bfix(f \mid g)(x)$*.*

Let fix_p^1, fix_i^1, $bfix_p^1$, $bfix_i^1$ stand for the fixpoint constructions in which the type τ is restricted to having the form $\{t\}$ (i.e. they allow fixpoints over only one set, not over several sets simultaneously). They are as expressive as fix_p, fix_i, $bfix_p$ and $bfix_i$ respectively (this is a straightforward generalization of the proof for flat relations, see e.g. [15]). But, in the presence of complex objects types, we can prove more:

Proposition 2 *Let* $\Omega_\sigma : unit \to \sigma$ *be the (totally) undefined function, and* $\Omega = \{\Omega_\sigma\}_{\sigma \in Types}$*, i.e. we have a symbol for the undefined object at each type* σ *(including complex types). Then*[5]*:* $\mathcal{NRA}_k(\Sigma) + fix_i^1 \subseteq \mathcal{NRA}_k(\Sigma) + fix_p \subseteq \mathcal{NRA}_{k+1}(\Sigma \cup \Omega) + fix_i^1$*. As a consequence, by extending* $\mathcal{NRA}(\Sigma)$ *with* any *of* fix_i*,* fix_p*,* fix_i^1 *or* fix_p^1 *we get, essentially, the same expressive power. Therefore, we shall abbreviate with* $\mathcal{NRA}(\Sigma) + fix$ *any of them.*

This proposition is in contrast to Abiteboul and Vianu's result in [7], that the inflationary and partial fixpoint have the same expressive power over first order logic iff $PTime = PSpace$.

Proof (Sketch) The partial fixpoint can express the inflationary fixpoint, because $fix_i(f) = fix_p(\lambda(x, y).y \cup f(x, y))$: so $\mathcal{NRA}_k(\Sigma) + fix_i^1 \subseteq \mathcal{NRA}_k(\Sigma) + fix_p^1$. Obviously, $\mathcal{NRA}_k(\Sigma) + fix_p^1 \subseteq \mathcal{NRA}_k(\Sigma) + fix_p$. So we only have to prove that $\mathcal{NRA}_k(\Sigma) + fix_p \subseteq \mathcal{NRA}_{k+1}(\Sigma) + fix_i^1$. Let $f : \sigma \times \tau \to \tau$. The idea for computing $fix_p(f)$, is to compute iteratively the sets $Y_n = \{y_0, y_1, \ldots, y_{n-1}\}$, where $y_0 = \emptyset$, $y_{k+1} = f(x, y_k)$. For this, define $g : \sigma \times \{\tau\} \to \{\tau\}$ to be: $g(x, Y) = \{\emptyset\} \cup (map\ \lambda y.f(x, y))(Y)$. Now if f has a partial fixpoint $y_n = fix_p(f)(x)$, then g has an inflationary fixpoint, namely $fix_i^1(g)(x) = \{y_0, y_1, \ldots, y_n\}$. So we start by computing $Y = fix_i^1(g)$. Next, we extract y_n from the set Y, by noticing that y_n is the only object $y \in Y$ having the property that $f(x, y) = y$. The problem is that sometimes $fix_i^1(g)(x)$ exists while $fix_p(f)(x)$ does not. E.g. when $y_4 = y_2$, then $fix_i^1(g)(x) = \{y_0, y_1, y_2, y_3\}$, while $fix_p(f)(x) =$ undefined. In this case, there is no $y \in Y$ s.t. $f(x, y) = y$, and, in order to simulate $fix_p(f)$, we have to return an undefined value - hence the necessity of having the Ω's in the language. So define $(fix_p\ f)(x) = if\ (\exists y \in fix_i^1(g)(x).(f(x, y) = y))\ then\ y\ else\ \Omega$[6]. Returning the unique y with the property that $f(x, y) = y$ may seem a problem: we can compute the singleton set $\{y\}$, but in general there is no way to extract the unique element from a set. In our case this *is* possible, because the type of

[5]For the language $\mathcal{NRA}_k(\Sigma \cup \Omega)$, we consider the conditional *if then else* to be nonstrict in the last two arguments

[6]In the calculus-style language, Ω_σ is just a constant of type σ, instead of a function of type $unit \to \sigma$.

y is a PS type. E.g. when $\tau = \{\tau_1\} \times \{\tau_2\}$, take $y = (\mu(\Pi_1(\{y\})), \mu(\Pi_2(\{y\})))$, etc. $\Box$

In particular, the above proof also implies that all total functions expressible in $\mathcal{NRA}(\Sigma) + fix_p$ are expressible in $\mathcal{NRA}(\Sigma) + fix_i^1$ as well, and that, by extending the definition of the partial fixpoint to a "total" fixpoint in a reasonable way, say by: $fix_t(f)(x) \stackrel{\text{def}}{=} y$ when $fix_p(f)(x)$ is defined and equal to y, and $fix_t(f)(x) \stackrel{\text{def}}{=} \emptyset$ otherwise, we get that $\mathcal{NRA}(\Sigma) + fix_t = \mathcal{NRA}(\Sigma) + fix_i$.

Example 2 *We present three ways to compute the powerset using the (unbounded) fixpoint. Consider the following functions* $f_i : \{\sigma\} \times \{\{\sigma\}\} \to \{\{\sigma\}\}$, $i = 1, 2, 3$*:* $f_1(x, Y) \stackrel{\text{def}}{=} \{\emptyset\} \cup map(\eta)(x) \cup map(\lambda(y_1, y_2).y_1 \cup y_2)(Y \bowtie Y)$, $f_2(x, Y) \stackrel{\text{def}}{=} \{\emptyset\} \cup map(\lambda(e, y).ins(e, y))(x \bowtie Y)$*, and* $f_3(x, Y) \stackrel{\text{def}}{=} \{x\} \cup map(\lambda(e, y).\, del(e, y))(x \bowtie Y)$ *(where* $ins, del : \sigma \times \{\sigma\} \to \{\sigma\}$ *are:* $ins(e, x) \stackrel{\text{def}}{=} \{e\} \cup x$, $del(e, x) \stackrel{\text{def}}{=} x - \{e\}$*). Then* $fix(f_i) : \{\sigma\} \to \{\{\sigma\}\}$ *computes the powerset, for all* $i = 1, 2, 3$*. In the same spirit, we can express the function* $even : \{\sigma\} \to \{unit\}$*, with* $even(x) = true$ *iff* $card(x)$ *is even, by first computing, using one of the above methods, the function* $p : \{\sigma\} \to \{\{\sigma\} \times \{unit\}\}$*, defined by* $p(x) = \{(y, even(y)) \mid y \subseteq x\}$*. In conjunction with proposition 5, this example shows that the unbounded fixpoint is strictly more powerful at higher types that the bounded fixpoint, even when* $\Sigma = \emptyset$*.*

In fact, when $\Sigma = \emptyset$, fixpoints are as powerful as powerset. This can be shown using example 2 and the idea from the proof of proposition 2.

Proposition 3 $\mathcal{NRA} + fix = \mathcal{NRA} + powerset$*.*

Unbounded fixpoints are known to express all "elementary queries":

Proposition 4 *(See [12])* $\mathcal{NRA} + fix =$ *the class of all Kalmar elementary queries, i.e. whose time - or space - complexity is* $exp(k, n)$ *(for some* k*), where* $n =$ *the size of the input, and* $exp(0, n) = n$, $exp(k+1, n) = 2^{exp(k,n)}$*. More, when* $k \geq 1$, $\mathcal{NRA}_{k+1} + fix_i$ *coincides with all queries whose* time *complexity is* $O(exp(k, n^c))$ *(for some* $c > 0$*), while* $\mathcal{NRA}_{k+1} + fix_p$ *coincides with the queries whose* space *complexity is* $O(exp(k, n^c))$ *(for some* $c > 0$*).*

The condition $\Sigma = \emptyset$ is important. Example 3 below shows that, for a certain Σ, $\mathcal{NRA}(\Sigma) + fix$ can express all recursive queries.

Example 3 *Let* Nat *be the signature containing the type* nat *of natural numbers, and the function symbols* $0 : unit \to nat$, $succ : nat \to nat$*. Let* $p : nat \times nat \to \{unit\}$ *be some predicate expressible in* $\mathcal{NRA}(\Sigma) + fix$*, and* $\varphi : nat \to nat$ *be the partial function for which* $\varphi(x) =$ *the smallest number* y *satisfying* $p(x, y) = true$*, i.e.* $\varphi(x) \stackrel{\text{def}}{=} min\{y \mid p(x, y)\}$*. The function* $F(x) = \{0, 1, \ldots, \varphi(x)\}$ *is expressible in* $\mathcal{NRA}(Nat) + fix$*, as* $F(x) \stackrel{\text{def}}{=} fix(f)(x)$*, where* $f(x, y) = if\ ext(\lambda z.p(x, z))(y)\ then\ y\ else\ \{0\} \cup map(succ)(y)$*. Then, we can* *express* $\eta \circ \varphi : nat \rightarrow \{nat\}$ *as* $(\eta \circ \varphi)(x) \stackrel{\text{def}}{=} \{y \mid y \in F(x) \text{ and } succ(y) \notin F(x)\}$. *This shows that the class of functions* $\varphi : nat \rightarrow nat$ *expressible in* $\mathcal{NRA}(Nat)+fix$ *is closed under the operation of minimization. In fact, we can prove that for any recursive* φ, *the function* $\eta \circ \varphi$ *is expressible in* $\mathcal{NRA}(Nat)+fix$. *This example shows that (1) for certain* Σ, $\mathcal{NRA}(\Sigma) + fix$ *can express more than just elementary queries, and (2) for certain* Σ, $\mathcal{NRA}_1(\Sigma) + fix$ *is more powerful than* $\mathcal{NRA}_1(\Sigma) + bfix$ *(see proposition 5 and lemma 1).*

For the bounded fixpoints, we have immediately:

Proposition 5 $\mathcal{NRA}(\Sigma) + bfix_p \subseteq PSpace$ *and* $\mathcal{NRA}(\Sigma) + bfix_i \subseteq PTime$, *provided that all functions in* Σ *are in* $PSpace$ *or* $PTime$ *respectively.*

Note that, in contrast to Proposition 4, this result holds for any Σ.

4 Main results

The main result of this paper is the following theorem (an extension of theorem 1), and in its corollary:

Theorem 2 *If all function symbols in* Σ *have set height* ≤ 1, *then*

1. $\mathcal{NRA}(\Sigma)+bfix$ *is a conservative extension of* $\mathcal{NRA}_1(\Sigma)+bfix$, *for both inflationary and partial fixpoint semantics. The latter is a conservative extension of the relational algebra with bounded fixpoints* $\mathcal{RA}(\Sigma) + bfix$.
2. *If* I *is an index type (to be defined), then* $\mathcal{NRA}(\Sigma \cup I) + fix$ *is a conservative extension of* $\mathcal{NRA}_1(\Sigma \cup I) + fix$, *which is, in turn, a conservative extension of* $\mathcal{RA}(\Sigma \cup I) + fix$.

The proof is given in sections 6 and 7, and consists essentially in encoding complex objects into flat relations with indexes, and in translating queries over complex objects into queries in the relational algebra extended with a function *pair* defined on the indexes. The translation also suggests a compilation method of complex objects into flat relations. The importance of the second part of the theorem lies in the fact that it shows that this compilation method works for queries with fixpoints as well as for queries with bounded fixpoints.

When $\Sigma = \emptyset$, then the bounded fixpoints and the unbounded fixpoint have the same expressive power over flat relations. More precisely:

Lemma 1 $\mathcal{NRA}_1+fix_p = \mathcal{NRA}_1+bfix_p$ *and* $\mathcal{NRA}_1+fix_i = \mathcal{NRA}_1+bfix_i$.

As we show in example 3 above, the condition $\Sigma = \emptyset$ is essential for this lemma.

Corollary 1 *1.* $\mathcal{NRA} + bfix_p$ *is a conservative extension of the relational algebra + partial fixpoint, which is equivalent to the while - queries ([11]), and to* $DATALOG^{*\neg}$ *([6]).*

2. *$\mathcal{NRA} + bfix_i$ is a conservative extension of the relational algebra + inflationary fixpoint which is equivalent to $DATALOG^{\neg}$, and to the first order logic with (inflationary or monotone) fixpoints ([4], [6]).*

This result has as a negative consequence the fact that the inclusions from prop 5 are strict: the function $even : \{\sigma\} \rightarrow \{unit\}$ which is definable with fix by proposition 4, is not expressible with bounded fixpoints. On the other hand, no database query language capable of capturing *exactly* the $PTime$ queries is currently known, so $bfix_i$ and $bfix_p$ seem to offer a reasonable extension to higher types, of the fixpoints in the relational algebra (or, equivalently, first order logic). As in the flat case, we have:

Theorem 3 *$\mathcal{NRA} + bfix_p + order = PSpace$ and $\mathcal{NRA} + bfix_i + order = PTime$.*

The proof goes along the lines of the proofs in [12] and [15], and is omitted.

5 A logic based language

We have two reasons for giving a logic based version of $\mathcal{NRA}(\Sigma) + bfix$. First, we want to gain a greater confidence in its robustness, and secondly, we want to understand how the bounded fixpoint can be expressed through range restriction.

We call our language $\mathcal{NRCALC}(\Sigma)+bfix$. It is strongly inspired by previous work ([1], [12], [13]), but its syntax is kept closer to the algebraic language, to simplify the proof of the conversions. Its main interest consists in the rules for range restriction, especially those connected to the fixpoint construction.

We consider a signature Σ given, as for $\mathcal{NRA}$. There is exactly one **input variable** for each type σ, x^σ, and there are denumerable many **variables** $u^\sigma, v^\sigma, \ldots$ for each type σ. The terms (t) and formulas (φ) are typed, and are defined inductively:

terms The following are terms: $()$ (the empty tuple), x^σ, $u^\sigma, v^\sigma, w^\sigma, \ldots$ (variables), (t_1, t_2) (the tuple), $f(t)$ (where $f \in \Sigma$ is a function symbol), $\pi_i(t)$ (the i's projection, $i = 1, 2$), $\{u^\tau \mid \varphi\}$ where u^τ is a variable and φ is some formula (the type of this term is $\{\tau\}$). u^τ becomes bound in $\{u^\tau \mid \varphi\}$.

formulas The following are formulas: $true$, $false$, $t_1 \in t_2$, $t_1 = t_2$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$ (where φ, ψ are formulas and t_1, t_2 are terms), $\exists u^\sigma \in t.\varphi$ (where u^σ is a variable, t is a term of type $\{\sigma\}$, and φ is some formula).

We extend the language with a fixpoint construction:

fixpoint $\mu w^{\{\tau\}}.\{u^\tau \mid \varphi\}$ is a term of type $\{\tau\}$, where $w^{\{\tau\}}$ and u^τ are variables, and φ is a formula. Both $w^{\{\tau\}}$ and u^τ become bound in $\mu w^{\{\tau\}}.\{u^\tau \mid \varphi\}$.

We shall assume that all bounded variables in some term t or formula φ are distinct, and distinct from the free variables[7].

A **query** of type $\sigma \to \tau$ in $\mathcal{NRCALC}(\Sigma)$ or $\mathcal{NRCALC}(\Sigma) + fix$, is simply a term of type τ, having only one free variable, namely x^σ, the input variable of type σ. Note that, although we can write some purely algebraic queries, like $(f(\pi_1(x), \pi_2(x)), g(\pi_2(x)))$, for "real" database queries one has to make use of formulas, like in $unnest : \{\sigma \times \{\tau\}\} \to \{\sigma \times \tau\}$: $unnest \stackrel{\text{def}}{=} \{u \mid \exists v \in x. \exists w \in \pi_2(v). u = (\pi_1(v), w)\}$, or in the following $nest$ query, $\{\sigma \times \tau\} \to \{\sigma \times \{\tau\}\}$, $nest \stackrel{\text{def}}{=} \{u \mid \exists v \in x. u = (\pi_1(v), \{w \mid (\pi_1(v), w) \in x\})\}$.

The values of terms and formulas depend on the values assigned to their free variables. Formulas can be $true$, $false$, or undefined. $\exists u \in t.\varphi$, is interpreted as follows: it is defined iff t is defined, and for all $a \in t$, $\varphi[u/a]$ is defined; in that case, $\exists u \in t.\varphi = true$ iff there is some $a \in t$ s.t. $\varphi[u/a] = true$. The interpretation of the term $\{u \mid \varphi\}$ is the set $\{\alpha \mid \varphi[u/\alpha] = true\}$; it is undefined, if the latter set is infinite[8]. For the fixpoint construction, we consider both inflationary and partial interpretations.

To avoid exponential queries, like the powerset of $x^{\{\sigma\}}$, $\{u^{\{\sigma\}} \mid \forall v^\sigma \in u^{\{\sigma\}}.v^\sigma \in x^{\{\sigma\}}\}$ ($\forall$ is expressed using $\exists$ and $\neg$), or domain dependent queries, like $\{u^\sigma \mid \neg u^\sigma \in x^{\{\sigma\}}\}$, we restrict the language, to the *range restricted* sublanguage $(\mathcal{NRCALC}(\Sigma) + bfix)^{rr}$. For this, let M be a term or a formula: we define certain subterm occurrences t to be range restricted in M - and then we write $\underline{t}$ -, and certain variables v to be range restricted in a subterm or subformula occurrences N of M - then we write $\underbrace{N}_{v}$. The following rules define the range restriction. Recall that the range restriction is a property of a subterm or subformula occurrence N, of some term or formula M.

$$\frac{op(\underline{t_1}, \ldots, \underline{t_m})}{\underline{op(t_1, \ldots, t_m)}} \text{(op is any operation)} \qquad \frac{\underline{(t_1, t_2)}}{(\underline{t_1}, \underline{t_2})} \qquad \frac{}{\underline{()}}$$

$$\underline{x} \text{(x is the input variable)} \qquad \frac{t_1 \in \underline{t_2}}{\underline{t_1} \in \underline{t_2}} \qquad \frac{\underline{t_1} = t_2}{\underline{t_1} = \underline{t_2}} \qquad \frac{t_1 = \underline{t_2}}{\underline{t_1} = \underline{t_2}}$$

$$\frac{\underline{u}}{\underbrace{u}_{u}} \qquad \frac{\overbrace{u}^{u}}{\underline{u}} \qquad \frac{}{\underbrace{false}_{u}}$$

$$\frac{\underbrace{op(t_1, \ldots, t_m)}_{u}}{op(\underbrace{t_1}_{u}, \ldots, \underbrace{t_m}_{u})} \qquad \frac{op(t_1, \ldots, \underbrace{t_i}_{u}, \ldots, t_m)}{\underbrace{op(t_1, \ldots, t_m)}_{u}}$$

[7] Any term or formula can be converted to an equivalent form, which satisfies this requirement, by renaming the bound variables.

[8] This will be impossible in the range restricted version of $\mathcal{NRCALC}$, to be defined below.

$$\frac{\underbrace{t_1}_{u} \in t_2}{\underbrace{t_1 \in t_2}_{u}} \qquad \frac{\underbrace{t_1 \in t_2}_{u}}{\underbrace{t_1}_{u} \in \underbrace{t_2}_{u}}$$

$$\frac{\underbrace{t_1}_{u} = t_2}{\underbrace{t_1 = t_2}_{u}} \qquad \frac{t_1 = \underbrace{t_2}_{u}}{\underbrace{t_1 = t_2}_{u}} \qquad \frac{\underbrace{t_1 = t_2}_{u}}{\underbrace{t_1}_{u} = \underbrace{t_2}_{u}}$$

$$\frac{\{u \mid \underbrace{\varphi}_{u}\}}{\underline{\{u \mid \varphi\}}} \qquad \frac{\mu w.\{u \mid \underbrace{\varphi}_{u}\}}{\underline{\mu w.\{u \mid \varphi\}}}$$

$$\frac{\exists u \in \underline{t}.\varphi}{\exists u \in \underline{t}.\underbrace{\varphi}_{u}} \qquad \frac{\exists u \in t.\underbrace{\varphi}_{v}}{\underbrace{\exists u \in t.\varphi}_{v}} \qquad \frac{\underbrace{\exists u \in t.\varphi}_{v}}{\exists u \in t.\underbrace{\varphi}_{v}}$$

$$\frac{\underbrace{\varphi}_{u} \wedge \psi}{\underbrace{\varphi \wedge \psi}_{u}} \qquad \frac{\varphi \wedge \underbrace{\psi}_{u}}{\underbrace{\varphi \wedge \psi}_{u}} \qquad \frac{\underbrace{\varphi \wedge \psi}_{u}}{\underbrace{\varphi}_{u} \wedge \underbrace{\psi}_{u}}$$

$$\frac{\underbrace{\varphi}_{u} \vee \underbrace{\psi}_{u}}{\underbrace{\varphi \vee \psi}_{u}} \qquad \frac{\underbrace{\varphi \vee \psi}_{u}}{\underbrace{\varphi}_{u} \vee \underbrace{\psi}_{u}} \qquad \frac{\underbrace{\neg\varphi}_{u}}{\neg \underbrace{\varphi}_{u}}$$

Note that, when a variable u is range restricted in the subformula φ of $\varphi \wedge \psi$, then it is range restricted also in ψ; this does *not* hold for $\vee$. Also, note the rule for the fixpoint (μ): we must be able to prove that u is range restricted, without assuming anything about the fixpoint variable w (which, in fact, may not be range restricted). This is a different approach than in [12].

A query is **range restricted** if it is range restricted as a term, and if all subterms occurring as bounds for quantifiers are range restricted (i.e. the quantifiers are of the form $\exists u \in \underline{t}.\varphi$). The **range restricted** fragment $(\mathcal{NRCALC}(\Sigma) + fix)^{rr}$ of $\mathcal{NRCALC}(\Sigma) + fix$ is defined to contain only the range restricted queries.

Example 4 *nest defined above is range restricted. Indeed:*

$$\{u \mid \exists v \in \underline{x}.u = (\pi_1(v), \{w \mid (\pi_1(v), w) \in \underline{x}\})\}$$

$$\{u \mid \exists v \in x.\underbrace{u = (\pi_1(v), \{w \mid (\pi_1(v), w) \in \underline{x}\})}_{v}\}$$

$$\{u \mid \exists v \in x.u = \underbrace{(\pi_1(v), \{w \mid (\pi_1(v), w) \in \underline{x}\})}_{v}\}$$

$$\{u \mid \exists v \in x.u = (\underbrace{\pi_1(v)}_{v}, \{w \mid (\pi_1(v), w) \in \underline{x}\})\}$$

$$\{u \mid \exists v \in x.u = (\pi_1(\underbrace{v}_{v}), \{w \mid (\pi_1(v), w) \in \underline{x}\})\}$$

$$\{u \mid \exists v \in x.u = (\pi_1(\underline{v}), \{w \mid (\pi_1(v), w) \in \underline{x}\})\}$$

$$\{u \mid \exists v \in x.u = (\underline{\pi_1(v)}, \{w \mid \underline{(\pi_1(v), w)} \in x\})\}$$

$$\{u \mid \exists v \in x.u = (\underline{\pi_1(v)}, \{w \mid (\pi_1(v), \underline{w}) \in x\})\}$$

$$\{u \mid \exists v \in x.u = (\underline{\pi_1(v)}, \{w \mid (\pi_1(v), \underbrace{w}_{w}) \in x\})\}$$

$$\{u \mid \exists v \in x.u = (\underline{\pi_1(v)}, \{w \mid \underbrace{(\pi_1(v), w) \in x}_{w}\})\}$$

$$\{u \mid \exists v \in x.u = (\underline{\pi_1(v)}, \underline{\{w \mid (\pi_1(v), w) \in x\}})\}$$

$$\{u \mid \exists v \in x.\underline{u} = (\pi_1(v), \{w \mid (\pi_1(v), w) \in x\})\}$$

$$\{u \mid \exists v \in x.\underbrace{u}_{u} = (\pi_1(v), \{w \mid (\pi_1(v), w) \in x\})\}$$

$$\{u \mid \underbrace{\exists v \in x.u = (\pi_1(v), \{w \mid (\pi_1(v), w) \in x\})}_{u}\}$$

$$\underline{\{u \mid \exists v \in x.u = (\pi_1(v), \{w \mid (\pi_1(v), w) \in x\})\}}$$

As this example shows, proving range restriction of some term requires us to apply several rules on subterms/subformulas of the same *term or formula.*

Theorem 4 *$(\mathcal{NRCALC}(\Sigma) + fix)^{rr}$ and $\mathcal{NRA}(\Sigma) + bfix$ have the same expressive power.*

6 Encoding of $\mathcal{NRA}(\Sigma)$ $(+fix, +bfix)$ into the relational algebra with indexes

In this section, we present the technique used for proving the conservativity result (theorem 2). The idea is to encode $\mathcal{NRA}(\Sigma) + bfix$ into the relational algebra over the same signature Σ plus indexes: $\mathcal{RA}(\Sigma \cup I) + bfix$.

First we consider the case when all function symbols in Σ have the set height 0. That is, Σ may contain functions like $add : nat \times nat \to nat$ or $zero : unit \to nat$, but not functions like $card : \{b\} \to nat$ or $sum : \{nat\} \to$

nat. Later we show how to extend this result to the case when Σ may contain functions of set height ≤ 1. We define the **the relational algebra** $\mathcal{RA}(\Sigma)$ to be a minor extension of the traditional relational algebra, as presented in [19]: its types (called **flat types**) are $\{t\}$ (with $sh(t) = 0$), and $\sigma \times \tau$ (with σ, τ flat types), and its operations contain union ($\cup$), difference ($-$), cartesian product ($\bowtie$), all functions of the form $map(f)$ with f in $\mathcal{NRA}_0(\Sigma)$[9], and all selection functions $\sigma_p : \{t\} \to \{t\}$, where $p : t \to \{unit\}$ is in $\mathcal{NRA}_0(\Sigma)$[10]. In addition, $\mathcal{RA}(\Sigma)$ contains the empty set $\emptyset : \{t\} \to \{t'\}$[11], and not. $\mathcal{RA}(\Sigma) + fix$ and $\mathcal{RA}(\Sigma) + bfix$ are extensions obtained by adding fix or $bfix$.

An **index** set is a base type I together with an injective function $pair : I \times I \to I$ (so I is infinite), and two distinct constants $left, right : unit \to I$. We write $\Sigma \cup I$ for the signature Σ extended with I, $left$, $right$ and $pair$.

The translation of $\mathcal{NRA}(\Sigma)$ ($+fix$ or $+bfix$) into $\mathcal{RA}(\Sigma \cup I)$ ($+fix$ or $+bfix$), begins with the translation of types: each type σ in $\mathcal{NRA}$ is translated into a flat type π_σ in $\mathcal{RA}$. Then, we define an *encoding relation* for each type: $\sim_\sigma \subseteq \sigma \times \pi_\sigma$. When $x \in \sigma$ and $r \in \pi_\sigma$, then $x \sim_\sigma r$ means that "r is an encoding of x": each x has at least one encoding (but it may have several), and it turns out that not every $r \in \pi_\sigma$ is somebody's encoding. However, the encoding is injective ($x \sim_\sigma r$, $x' \sim_\sigma r \Rightarrow x = x'$), and the decoding function $decode_\sigma : \pi_\sigma \to \sigma$ is also injective, expressible in $\mathcal{NRA}$, and is even in $\mathcal{RA}$, when σ is flat. Finally, we define a translation of some function $f : \sigma \to \tau$ in $\mathcal{NRA}(\Sigma)$ ($+fix$ or $+bfix$) into a function $R_f : \pi_\sigma \to \pi_\tau$ in $\mathcal{RA}(\Sigma \cup I) + bfix$, such that the following **soundness** property holds: $x \sim_\sigma r \Rightarrow f(x) \sim_\tau R_f(r)$.

The translation of types and the encoding relation are: $\pi_b \stackrel{\text{def}}{=} \{b\}$ and $\forall x \in b, x \sim_b \{x\}$, $\pi_{\sigma\times\tau} \stackrel{\text{def}}{=} \pi_\sigma \times \pi_\tau$ and $(x, y) \sim_{\sigma\times\tau} (r, q)$ iff $x \sim_\sigma r$ and $y \sim_\tau q$. The difficult part is encoding $\{\sigma\}$. Recall that π_σ is a flat type, and we need to encode finite sets r from π_σ (i.e. $r \in \{\pi_\sigma\}$) also as elements of some flat type. The idea is to encode r as a partial function $\psi : I \Rightarrow \pi_\sigma$ having the range r. The function is *finite*, i.e. it is defined only on a finite subset of I. Suppose π_σ is a set type, say $\pi_\sigma = \{t\}$. A straightforward representation of such a function is as a binary relation of type $\{I \times \{t\}\}$. However, this relation is not flat, and we need a more sophisticated approach, namely we unnest the above relation, to get a relation of type $\{I \times t\}$, and denote this type by $[I \to \pi_\sigma]$. Any relation $R \in [I \to \pi_\sigma]$ encodes some partial, finite function $\psi : I \to \pi_\sigma$, namely $\forall i \in \Pi_1(R)$, $\psi(i) \stackrel{\text{def}}{=} \Pi_2(\sigma_{1=i}(R))$, and $\forall i \in I - \Pi_1(R)$, $\psi(i) =$ undefined. This is almost what we need, except that we cannot encode functions ψ for which for some $i \in I$, $\psi(i)$ is defined and is $\emptyset$. To take this into account, let $dom(\psi)$ be the finite domain of ψ. We represent ψ by $(dom(\psi), R)$, a relation of type $\{I\} \times [I \to \pi_\sigma]$: we denote this type by $[I \Rightarrow \pi_\sigma]$. Any finite, partial function $\psi : I \to \pi_\sigma$ has a representation as a pair of flat relations of type $[I \Rightarrow \pi_\sigma]$,

[9] These are generalized projections. E.g. from $\pi_1 : t_1 \times t_2 \to t_1$ is in $\mathcal{NRA}_0(\Sigma)$, we get $\Pi_1 \stackrel{\text{def}}{=} map(\pi_1)$ in $\mathcal{RA}(\Sigma)$.

[10] $\sigma_p \stackrel{\text{def}}{=} ext(\Pi_2 \circ \bowtie \circ (p, \eta))$. E.g. from $p : (b \times b) \times t \to \{unit\}$, $p = eq \circ \pi_1$, we get the selection σ_p, traditionally written as $\sigma_{1=2}$.

[11] Recall that the type $unit$ is not in $\mathcal{RA}(\Sigma)$.

and any such pair (D,R) with the property that $\Pi_1(R) \subseteq D$ represents some function ψ.

Formally, for any flat type $\nu = \{t_1\} \times \ldots \times \{t_n\}$, we define $[I \to \nu] \stackrel{\text{def}}{=} \{I \times t_1\} \times \ldots \times \{I \times t_n\}$ and $[I \Rightarrow \nu] \stackrel{\text{def}}{=} \{I\} \times [I \to \nu]$. There is a one to one correspondence between the partial, finite, *nonempty*[12] functions $\psi : I \to \nu$ and the elements of type $[I \to \nu]$, and there is a one to one correspondence between the partial finite functions $\psi : I \to \nu$ and those values $(D, R_1, \ldots, R_n)$ of type $[I \Rightarrow \nu]$ which satisfy $\Pi_1(R_1) \subseteq D, \ldots, \Pi_1(R_n) \subseteq D$: for some partial finite function ψ, we denote $rel(\psi)$ its encoding in $[I \Rightarrow \nu]$.

Returning to our translation of types, we define $\pi_{\{\sigma\}} \stackrel{\text{def}}{=} [I \Rightarrow \pi_\sigma]$ and $x \sim_{\{\sigma\}} r$ iff there are two finite, partial functions $\omega : I \Rightarrow \sigma$ and $\psi : I \Rightarrow \pi_\sigma$, such that $dom(\omega) = dom(\psi)$, $\forall i \in dom(\psi).\omega(i) \sim_\sigma \psi(i)$ and $x = range(\omega)$, $r = rel(\psi)$.

The encoding of the operations $f \rightsquigarrow R_f$ proceeds in two steps. First, one encodes all operations f from $\mathcal{NRA}(\Sigma)$ (i.e. without fixpoints). $\mu : \{\{\sigma\}\} \to \{\sigma\}$ is translated int $R_\mu : [I \Rightarrow [I \Rightarrow \sigma]] \to [I \Rightarrow \sigma]$ as follows: first one notes that $[I \Rightarrow [I \Rightarrow \sigma]] = \{I\} \times [I \times I \Rightarrow \sigma]$, so R_μ is defined using $pair : I \times I \to I$, which extends (via map) to a function $[pair \Rightarrow \sigma] : [I \times I \Rightarrow \sigma] \to [I \Rightarrow \sigma]$. The injectivity of $pair$ is used in the proof of the soundness of R_μ. Instead of $\cup$, we encode $doubleton$[13] $: \sigma \times \sigma \to \{\sigma\}$ by using $left$ and $right$, and use $left \neq right$ for soundness ($\cup$ is defined indirectly, using $doubleton$), $R_{map(f)} \stackrel{\text{def}}{=} [I \Rightarrow R_f]$. η is translated using $left$ (chosen arbitrarily from $left$ and $right$). Some function symbol $f : t_1 \times \ldots \times t_m \to t$ in Σ, is encoded by $R_f : \{t_1\} \times \ldots \times \{t_m\} \to \{t\}$, $R_f(r_1, \ldots, r_m) \stackrel{\text{def}}{=} map(f)(r_1 \bowtie \ldots \bowtie r_m)$. The rest of the cases are trivial.

Note that $R_\cup \neq \cup$ and $R_\cap \neq \cap$: both $\cup$ and $\cap$ are derived operations. More, even if σ is a set type and $r, q \in \pi_\sigma$ are valid encodings (of, say, x and y), it is not necessarily the case that $r \cup q$ is a valid encoding[14], because r and q might accidentally use the same index for encoding different elements. Therefore, the encoding of $bfix$ and fix has to be done with some care, and we use different techniques for each of them.

To encode $bfix(f,g)$, we proceed as follows. Suppose $f : \sigma \times \tau \to \tau$, $g : \sigma \to \tau$ (τ a PS type), and let R_f, R_g be their translations. Define $Q : \pi_\sigma \times \pi_\tau \to \pi_\tau$ as follows. To compute $Q(r,q)$, first compute $s \stackrel{\text{def}}{=} R_f(r,q)$ and $t \stackrel{\text{def}}{=} R_g(q)$. Next "decode" s, t to get $u, v \in \tau$. Intersect them, to get $w \stackrel{\text{def}}{=} u \cap v$, and encode w, using the indexes from t: all this can be done in $\mathcal{RA}$. We define the result to be $Q(r,q)$. So, $Q(r,q) \subseteq R_g(q)$ and, whenever $x \sim_\sigma r$, $y \sim_\tau q$, we have $f(x,y) \cap g(x) \sim_\tau Q(r,q)$. Finally, define $R_{bfix(f,g)} \stackrel{\text{def}}{=} bfix(Q, R_g)$. For the partial fixpoint, the soundness of the translation follows immediately, for the inflationary fixpoint we need to observe that $x \sim r$, $y \sim q$, $z \sim s$ and $x \subseteq z$, $y \subseteq z$, $r \subseteq s$, $q \subseteq s$ implies $x \cup y \sim r \cup q$.

[12]I.e. when $\psi(i)$ is defined, then $\psi(i) \neq (\emptyset, \ldots, \emptyset)$.

[13]$doubleton : \sigma \times \sigma \to \{\sigma\}$ is defined as $\cup \circ (\eta \times \eta)$, with the meaning $doubleton(x,y) = \{x,y\}$. Then $\cup \stackrel{\text{def}}{=} \mu \circ doubleton$.

[14]At first glance, one would expect $r \cup q$ to be an encoding of $x \cup y$.

To encode $fix(f)$, where $f : \sigma \times \tau \to \tau$, one defines $Q : \pi_\sigma \times \pi_\tau \to \pi_\tau$ as follows. First, compute $s \stackrel{\text{def}}{=} R_f(r,q)$, where R_f is the translation of f, and let s' be the set of "new" values, i.e. which were not in q: more precisely, $s' = R_-(s,q)$, where R_- is the translation of $-$ (minus). As for the bounded fixpoint, decode s and q to get $u, v \in \tau$, compute $w \stackrel{\text{def}}{=} u \cap v$, and encode w using the indexes from q: call the result q'. We would like to define $Q(r,q) \stackrel{\text{def}}{=} s' \cup q'$, but the latter is not necessarily a valid encoding, because s' and q' may accidentally use common indexes. So we repeatedly change all indexes used in s', by replacing each index i with $pair(left, i)$, until none of its indexes occurs in q': an unbounded fixpoint is needed at this stage, and we assume that $(I, pair)$ is freely generated, to make sure that this process halts. Call s'' the result (i.e. both s' and s'' encode the same value, but they may use different indexes). Then define $Q(r,q) \stackrel{\text{def}}{=} s'' \cup q'$. Clearly $R_f(r,q)$ and $Q(r,q)$ are encodings of the same value, but Q ensures that $f(x,y_n) \subseteq y_n$ implies $Q(r,q_n) \subseteq q_n$. Finally, define $R_{fix(f)} \stackrel{\text{def}}{=} fix(Q)$.

This construction is enough to prove that $\mathcal{NRA}(\Sigma \cup I) + fix$, and $\mathcal{NRA}(\Sigma \cup I) + bfix$, are conservative extensions of $\mathcal{RA}(\Sigma \cup I) + fix$, and $\mathcal{RA}(\Sigma \cup I) + bfix$ respectively, and, hence, that all expressions over flat types in $\mathcal{NRA}(\Sigma) + fix$ and $\mathcal{NRA}(\Sigma) + bfix$ are also in $\mathcal{RA}(\Sigma \cup I) + fix$ and $\mathcal{RA}(\Sigma \cup I) + bfix$ respectively. Using this result, it is straightforward to prove that $\mathcal{NRA}(\Sigma \cup I) + fix$ and $\mathcal{NRA}(\Sigma \cup I) + bfix$ are also conservative extensions of $\mathcal{NRA}_1(\Sigma \cup I) + fix$ and $\mathcal{NRA}_1(\Sigma \cup I) + bfix$ respectively. This concludes the proof of the part 2 of theorem 2. For the part 1 however, we need to eliminate the indexes from R_f, when $f : \sigma \to \tau$ is a function over flat types, which is done in the next section.

7 Elimination of indexes for encodings of flat relations

Let σ, τ be flat types, and $f : \sigma \to \tau$ be a function in $\mathcal{NRA}(\Sigma) + bfix$. Compose the function $R_f : \pi_\sigma \to \pi_\tau$ (which is in $\mathcal{RA}(\Sigma \cup I) + bfix$), with $decode_\tau : \pi_\tau \to \tau$ (which is expressible in $\mathcal{RA}$), to get $Q : \pi_\sigma \to \tau$, with the property $x \sim_\sigma r \Rightarrow f(x) = Q(r)$. Then, we use the following:

Lemma 2 *Let $Q : \pi_\sigma \to \tau$ be in $\mathcal{RA}(\Sigma \cup I) + bfix$, such that, for any index I, if $r, r' \in \pi_\sigma$ are encodings of the same $x \in \sigma$, then $Q(r) = Q(r')$. Then there is some function $g : \sigma \to \tau$ in $\mathcal{RA}(\Sigma) + bfix$ (without indexes), such that $x \sim_\sigma r \Rightarrow g(x) = Q(r)$.*

Proof (Sketch) For simplicity, assume $\sigma = \{t_1\} \times \{t_2\}$, where $t_1 = b$ and $t_2 = b \times b$: then $\pi_\sigma = (\{I\} \times \{I \times b\}) \times (\{I\} \times \{I \times b\} \times \{I \times b\})$. In general, there is no function $encode_I : \sigma \to \pi_\sigma$ in $\mathcal{RA}$, with the property: $x \sim_\sigma encode_I(x)$, $\forall x \in \sigma$, simply because such a function would not be *generic*. We consider a particular index set I, namely the smallest set containing the disjoint union

$unit_1 \sqcup unit_2 \sqcup t_1 \sqcup t_2$[15], and closed under pairing (i.e. $x, y \in I \Rightarrow (x, y) \in I$). Define $pair(x, y) \stackrel{\text{def}}{=} (x, y)$, $left \stackrel{\text{def}}{=} ()_1$ and $right \stackrel{\text{def}}{=} ()_2$. Then t_1 and t_2 are subtypes of I, so $(\{t_1\} \times \{t_1 \times b\}) \times (\{t_2\} \times \{t_2 \times b\} \times \{t_2 \times b\}) \subseteq \pi_\sigma$, and $\mathcal{RA}$ can express some function $encode : \sigma \to (\{t_1\} \times \{t_1 \times b\}) \times (\{t_2\} \times \{t_2 \times b\} \times \{t_2 \times b\})$ with the above property (but, of course, with a different type). As defined, I is a disjoint union of scalar types. Now, we look at the syntactic structure of the expression $Q(r)$ as a tree, and label its nodes with their types. In this tree, we replace the lowest occurrences of I (namely in the type π_σ of the input variable r), with the subtypes t_1 and t_2 respectively (π_σ becomes now $(\{t_1\} \times \{t_1 \times b\}) \times (\{t_2\} \times \{t_2 \times b\} \times \{t_2 \times b\})$). Step by step, we transform the tree $Q(r)$ bottom up, into an equivalent expression, such that any occurrence of the type I is replaced by some of its subtypes t. At the same time, each occurrence of the function $pair : I \times I \to I$, is replaced by the identity $id : t \times t' \to t \times t'$, for appropriate subtypes $t, t', t \times t'$ of I, while $left$ and $right$ are replaced by $()$. Call $\bar{Q}$ the resulting expression: then take $g(x) \stackrel{\text{def}}{=} \bar{Q}(encode(x))$. $\Box$

We end this section, with a comment about the case when Σ contains function symbols of set height 1, e.g. aggregate functions, like $sum : \{nat\} \to nat$, or generators like $gen : nat \to \{nat\}$ $(gen(n) \stackrel{\text{def}}{=} \{0, 1, \ldots, n\})$. Then, we proceed as above, with one exception: to define R_f for function symbols $f \in \Sigma$ whose range is a set, say $\{s\}$, we need index invention. So we add to the definition of an index I, a family of "invention" functions $c_s : \{s\} \to \{I \times s\}$, for each set type $\{s\}$ which is the range of some function symbol in Σ. These too can be eliminated, like in the proof of lemma 2, by imposing s to be a subtype of I, and replacing c_s with $map(id, id) : \{s\} \to \{s \times s\}$.

8 Concluding remarks and further research

We have investigated the power of the nested relational algebra enriched with a bounded fixpoint. The language turned out to be still of polynomial time or space complexity - according to the interpretation of the fixpoint: inflationary or partial - and it can express all *PTime* or *PSpace* queries over ordered databases. The main result consists in proving that, over flat relations, the language has the same expressive power as the first order logic with inflationary or partial fixpoints. We also define a logic based language, having the same expressive power.

The method used in the proof of the conservativity result could be used as a compilation technique, for encoding complex objects by flat relations, and for translating queries in $\mathcal{NRA} + bfix$ into queries of the relational algebra extended with a function *pair* and with bounded fixpoints.

Rather than having a fixed interpretation, the function *pair* is more like an oracle, which may be consulted when computing some query: this view is different from that of *interpreted* functions in [1]. By defining $succ(x) \stackrel{\text{def}}{=} pair(x, x)$ and $nat \stackrel{\text{def}}{=} \{left, succ(left), succ^2(left), \ldots\} \subseteq I$, we can express

[15] $unit_1 = unit_2 = unit$: we just want, for the moment, to distinguish the two copies in I.

in $\mathcal{NRA}(\Sigma \cup I) + fix$ all partial recursive functions over nat: however, unlike [1], the query $p(x) = \{y \mid pair(y, y) = x\}$ is not expressible in this language. We would like to investigate such "queries with oracles" in the future. For example, it might be useful to redefine the notion of a generic function, such that the p is no longer "generic"; we hope to be able to prove that all generic functions are expressible in $\mathcal{NRA}(\Sigma \cup I) + fix$. We also intend to compare the expressive power of our language with that of $RR - (CALC + IFP)$ and $RR - (CALC + PFP)$, in [12].

Finally, we believe that our methods can be extended to show that, for any $k \geq 1$, $\mathcal{NRA}(\Sigma) + bfix$ is a conservative extension of $\mathcal{NRA}_k(\Sigma) + bfix$, provided that all functions in Σ have set height $\leq k$.

9 Acknowledgements

I wish to give special thanks to Val Breazu-Tannen, for numerous discussions and comments, to Peter Buneman, for his suggestions and encouragement, and to Catriel Beeri, for his careful reading of this paper, and for his many comments and suggestions. I also wish to thank Leonid Libkin and Limsoon Wong, for their help and suggestions.

References

[1] S. Abiteboul, C. Beeri, *On the power of languages for the manipulation of complex objects*, Technical Report 846, INRIA, 1988

[2] S. Abiteboul, S. Grumbach, A. Voisard, E. Waller *An Extensible Rule-Based Language with Complex Objects and Data-Functions*, Proc. DBPL-II Workshop, Oregon, 1989

[3] S. Abiteboul, P. Kanellakis, *Object Identity as a Query Language Primitive*, Proc. of ACM SIGMOD conf, 1989

[4] S. Abiteboul, M. Vardi, V. Vianu, *Fixpoint Logics, Relational Machines, and Computational Complexity*, Proc. Conf. on Structure in Complexity Theory, 1992.

[5] S Abiteboul, V. Vianu, *Fixpoint extensions of first-order logic and Datalog-like languages*, Proc. 4th IEEE Symp. on Logic in Computer Science, pp 71-79, 1989

[6] S. Abiteboul, V. Vianu, *Expressive Power of Query Languages*, Theoretical Studies in Computer Science, ed. J. Ullman, Academic Press, 1991.

[7] S. Abiteboul, V Vianu, *Generic Computation and Its Complexity*, Proc ACM Symposium on Theory of Computing, 1991

[8] V. Breazu-Tannen, P. Buneman, S. Naqvi, *Structural Recursion As A Query Language*, MS-CIS-92-17, University of Pennsylvania

[9] V. Breazu-Tannen, P. Buneman, L. Wong, *Naturally Embedded Query Languages*, in Biskup and Hull, editors, LNCS 646: *Proceedings of the International Conference on Database Theory, Berlin, Germany*, October 1992, Springer-Verlag

[10] A. Chandra, D. Harel, *Computable Queries for Relational Data Bases*, Journal of Computer and System Sciences, 21:156-178, 1980

[11] A. Chandra, D. Harel, *Structure and complexity of relational queries*, Journal of Computer and System Sciences, 25:99-128, 1982

[12] S. Grumbach, V. Vianu, *Tractable Query Languages for Complex Object Databases*, Proc ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1991

[13] R. Hull, J. Su, *On the Expressive Power of Database Queries with Intermediate Types* Proc 7th Symp. on Principles of Database Systems, 1988

[14] R. Hull, J.Su, *Untyped Sets, Invention and Computable Queries* Proc 8th ACM Symp. on Principles of Database Systems, 1989

[15] N. Immerman, *Relational queries computable in polynomial time*, Information and Control, 68:86-104, 1986

[16] L. Libkin, L. Wong, *Aggregate Functions, Conservative Extension, and Linear Orders*, Proc. 4th International Workshop in Database Programming Languages, 1993 (this volume)

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

[18] Yatin Saraiya, *Fixpoints and optimizations in a language based on structural recursion on sets*, Manuscript, December 1992.

[19] J. D. Ullman, *Database and Knowledge-Base Systems*, Vol I and II, Computer Science Press, 1989

[20] J. Van den Bussche, *Complex Object Manipulation through Identifiers - an Algebraic Perspective*, TR 92-41, Universitaire Instelling Antwerpen, Sept. 1992

[21] M. Vardi, *The complexity of relational query languages*, Proc ACM SIGACT Symp. on the Theory of Computing, pp 137-146, 1982

[22] D. Van Gucht, P. Fischer *Multilevel Nested Relational Structures*, Journal of Computer and System Sciences 36, 77-105, 1988

[23] L. Wong, *Normal Forms and Conservative Properties for Query Languages over Collection Types*, to appear in PODS 93