March 1993

Fixpoints and Bounded Fixpoints for Complex Objects

Dan Suciu
University of Pennsylvania

Fixpoints and Bounded Fixpoints for Complex Objects

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 *NRA* with a "bounded fixpoint" operator. As in the flat case, 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 *NRA* with unbounded fixpoints into flat relations. We also define a logic based language with fixpoints, the "nested relational calculus", and prove that its range restricted version is equivalent to *NRA* with bounded fixpoints.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-93-32.

Fixpoints and Bounded Fixpoints for Complex Objects

Dan Suciu [*]
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389
Email: suciu@saul.cis.upenn.edu

March 1993

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 $\mathcal{NRA}$ with a "bounded fixpoint" operator. As in the flat case, 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 logic based language with fixpoints, the "nested relational calculus", and prove that its range restricted version is equivalent to $\mathcal{NRA}$ with bounded fixpoints.

1 Introduction

Several query languages for databases with complex objects have been studied lately ([1], [2], [3], [9], [12], [15], [16], [20], [23], [25]). The various systems investigated tend to share a common - or related - type system, allowing arbitrary nesting of cartesian products and finite-set constructions, starting from basic types. The query languages for these complex objects - like in the relational case - come in two different flavors: *logic* based languages, and *algebraic* based languages. Like in the relational case, these two turn out to be equivalent (Abiteboul and Beeri: ([1])).

These languages fall into two distinct equivalent classes, according to their expressive power. The first class is less expressive, and all its queries are in *PTime*: the *nested relational algebra* $\mathcal{NRA}$ in [9], or the logic based *restricted safe calculus*, in [1] and its equivalent *algebra without fixpoints* in [1] are representatives of this class. The main property of the languages in this class, is that they are an *conservative extension* of the relational algebra (see, e.g., [22]). This is a nontrivial property, proved first in [20], later (in a more general form) in [29], and also in [23]. As a consequence, complex objects cannot help us to compute more complicated queries at *flat types* (i.e. sets of products of base types): as a classic example, the transitive closure of a given relation cannot be computed in these languages, because it can't be computed in the relational algebra.

The languages in the second class have a richer expressive power: they are able to compute the powerset. Consequently, they can express queries whose time (or space) complexity is exponential (in fact, a tower

[*]The author was partially supported by grant NSF CCR-90-57570

of exponentials). $\mathcal{NRA} + powerset$ in [9], or the logic based *safe calculus* and the equivalent *algebra* (with powersets), from [1], or $CALC$, in [15], [12], are examples of languages in this class.. In [1], Abiteboul and Beeri prove that the powerset construction is equivalent to the *fixpoint* construction, and in [9], Breazu-Tannen, Buneman and Wong prove that, under certain conditions, it is equivalent to the *structural recursion*, a construction more in the style of functional programming. In contrast to the first category of languages, those in the second category are no longer conservative extensions of their first order cousins - first order logic with fixpoints, for example: using higher types as intermediate types, we can express queries over flat types which are not expressible in, say, FO+fixpoints. In fact, [15] prove that the expressive power of $CALC$ at base types becomes strictly more powerful as we allow intermediate types of larger set height. This is not surprisingly: queries in FO logic with fixpoints are of a low complexity class ($PTime$, for FO with *inflationary* fixpoints, and $PSpace$, for FO with *partial* fixpoints), while queries in $CALC$ (or $\mathcal{NRA} + powerset$) have a complexity which is a tower of exponentials.

In this paper, we design a language whose expressive power lies between the expressive powers of the two classes mentioned above: all queries it expresses are *tractable*, and it is an extension of the first order logic with fixpoints. The language is constructed by adding, to the *nested relational algebra* $\mathcal{NRA}$ of [9], the *bounded* fixpoint construction (an idea due to Peter Buneman, who also conjectured the conservativity result). When $f : \sigma \times \{\tau\} \to \{\tau\}$, then the **fixpoint** of f is $fix(f) : \sigma \to \{\tau\}$, with two alternative semantics: the **inflationary** semantics $fix_i(f)(x) = \bigcup_{n\geq 0} y_n$, where $y_0 = \phi$, $y_{n+1} = y_n \cup f(x, y_n)$, and the **partial** semantics $fix_p(f)(x) = y_k$, if $y_0 = \phi$, $y_{n+1} = f(x, y_n)$ and $y_k = y_{k+1}$. We shall give examples of $PTime$ functions f, for which, when x is some set, $fix_i(f)(x)$ is the powerset of x. The **bounded fixpoint** of f and some function $g : \sigma \to \{\tau\}$, is defined to be $bfix(f \mid g)(x) = \bigcup_{n\geq 0} y_n$, where $y_0 = \phi$, $y_{n+1} = y_n \cup f(x, y_n) \cap g(x)$. $bfix_i(f \mid g)$ is in $PTime$, when f and g are in $PTime$, while $bfix_p(f \mid g)$ is in $PSpace$, when f and g are in $PSpace$. Note that in the flat case, when f is a generic database transformation, then an upperbound for $fix(f)(x)$ can be easily computed from x, so any fixpoint can be expressed as a bounded fixpoint.

To give more confidence in the robustness of the resulting language, $\mathcal{NRA} + bfix$, we present a logic based equivalent, called $\mathcal{NRC} + fix$, which seems to be different from the range-restricted CALC in [12]. $\mathcal{NRA} + bfix$ shares a common property with the languages from the first category: it is a conservative extension of the first order logic with fixpoints. This has as immediate consequence the fact that $\mathcal{NRA}+bfix$ cannot express - at flat types - queries which were not expressible in FO logic with fixpoints (the classical example being the parity test). But, as in the flat case (see [24] and [17]), $\mathcal{NRA} + bfix$ can compute any $PTime$ or $PSpace$ database transformation on ordered databases.

The main result of this paper is the conservativity property of $\mathcal{NRA} + bfix$ over first order logic with fixpoints. For the proof, we use a different technique than the one used in [20] and [29]: we don't know how to adapt their reduction methods for the bounded fixpoint. We use a semantic approach instead, namely we encode all complex object types into flat types (i.e. relations), using indexes from some infinite set, say I. The fact that such an encoding is possible, is no mystery: this is the reason for which first order normal form for databases was advocated in the first place. What is less clear, are the operations one needs to add to the relational algebra $\mathcal{RA}$, in order to simulate all functions in $\mathcal{NRA}$ at higher types. In [23], where indexes were also used to prove a conservativity result, "index invention" is used. This is a nondeterministic operation, and is motivated by the need to encode the *nest* function, $nest : \{\sigma \times \tau\} \to \{\sigma \times \{\tau\}\}$, a primitive function in [23], which *creates* an arbitrary large number of new sets. Here, we take a different approach, namely we equip I with two constants, and a binary function $pair : I \times I \to I$, which we require to be injective.

Note that, in $\mathcal{RA}$ *with* an index I, the bounded fixpoint is no longer equivalent to the unbounded one, because the latter might produce infinite sets. $\mathcal{NRA} + fix$ can still be translated into the relational algebra with an index I, but with unbounded fixpoints. Only for $\mathcal{RA} + bfix$ with indexes can we prove that, in certain conditions, the indexes may be eliminated altogether, proving thus the conservativity result.

Another proposal for a tractable language for complex objects with fixpoints, was given in Grumbach and Vianu [12]. It is a logic based language, whose range-restriction rules enforce polynomial time behavior, even for the fixpoint. It is not clear to us whether this language is closed under *map* (following the terminology of [9] - which we adopt in this paper), or *replace* (ρ) (following the terminology of [1]), a property enjoyed by many complex-object query languages. One reason we are considering a logic based equivalent to our language, is to compare it with Grumbach and Vianu's. A problem in clarifying the relationship, is the difference in the range restriction rules. We hope to be able to clarify the relationship between them in the near future.

In section 2, we give the basic definitions of $\mathcal{NRA}$, and state the most important known results. In section 3, we define the unbounded and the bounded fixpoint, show that both semantics of the unbounded fixpoint (inflationary and partial) yield the same expressive power, and that the bounded fixpoints *capture*[1] the complexity classes *PTime* and *PSpace* respectively. Section 4 presents a logic based version of $\mathcal{NRA}+bfix$, called $\mathcal{NRC}+fix$, and prove their equivalence. Sections 5 to 9 are devoted to the proof of the main result (the conservativity theorem): section 5 describes the technique of encoding the set constructor in the relational algebra $\mathcal{RA}$, section 6 defines the translation from $\mathcal{NRA}$ to $\mathcal{RA}$ with indexes, sections 7 and 8 show how to eliminate the indexes, from certain expressions. The proof of the conservativity result is finally given in section 9. The proof, as given, works only for the case when, all functions in the signature Σ have set height 0: in section 10, we extend the proof to the case when Σ contains functions of set height ≤ 1.

2 The Nested Relational Algebra

Consider Σ to be a collection of basic types b, and function symbols $f : d_f \to c_f$, where d_f and c_f are *types* (to be formally defined below), called the *domain* and *codomain* of f. A set of typed *variables* $\mathcal{X}$ is also fixed: it contains countable many variable x^σ for each type σ.

Our core language is the **Nested Relational Algebra** $\mathcal{NRA}(\Sigma, \mathcal{X})$ ([9]). Its **types** are: the **base** types b from Σ, the **unit** type *unit*, the **product** type $\sigma \times \tau$, and the **set** type $\{\sigma\}$ (where σ, τ are types). The **operations** are described by the following rules:

$$\frac{f \in \Sigma}{f : d_f \to c_f} \qquad \frac{x^\sigma \in \mathcal{X}}{x^\sigma : unit \to \sigma}$$

$$\frac{}{id_\sigma : \sigma \to \sigma} \qquad \frac{f : \sigma \to \tau \quad g : \tau \to \nu}{g \circ f : \sigma \to \nu}$$

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

$$\frac{}{\eta_\sigma : \sigma \to \{\sigma\}} \qquad \frac{}{\mu_\sigma : \{\{\sigma\}\} \to \{\sigma\}}$$

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

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

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

[1]Following [4], a database language $\mathcal{L}$is said to *capture* some complexity class $\mathcal{C}$, if all queries in $\mathcal{L}$ are in $\mathcal{C}$, and $\mathcal{L}$ can express all database transformations in $\mathcal{C}$ on ordered databases.

The semantics of these operations is fully described in [9]. Here, we briefly sketch it: the type *unit* contains only $()$, the elements of the type $\{\sigma\}$ are finite sets of elements of type σ. $\pi_1(x,y) = x$, $(f_1, f_2)(x) = (f_1(x), f_2(x))$, $\iota_\sigma(x) = ()$, $\eta_\sigma(x) = \{x\}$, $\mu_\sigma(\{\{x,y\},\{z\}\}) = \{x,y,z\}$, $map(f)(\{x,y,z\}) = \{f(x), f(y), f(z)\}$, $\rho_{2,(\sigma,\tau)}(x, \{a,b,c\}) = \{(x,a),(x,b),(x,c)\}$, ϕ_σ returns the empty set, $\cup_\sigma$ is set union, $not(\phi) = \{()\}$ and $not(\{()\}) = \phi$, $eq_b(x,x) = \{()\}$ and $eq_b(x,y) = \phi$ when $x \neq y$.

Following [9], we impose equations on the syntactic constructs of the language, making it a category whose objects are the types, and whose morphisms are the operations of $\mathcal{NRA}(\Sigma, \mathcal{X})$. This category has binary products, a terminal object (*unit*), and is equipped with a **ringad** ([21]), i.e. a **monad** $(\{\ \}, \eta, \mu)$ (see [19]), together with a monoid structure $(\{\sigma\}, \cup_\sigma, \phi_\sigma)$ for each set type $\{\sigma\}$. The presence of $\rho_{2,(\sigma,\tau)}$ makes our monad a "strong monad" (see [9]).

From $\rho_{2,(\sigma,\tau)}$, one defines its symmetrical $\rho_{1,(\sigma,\tau)} : \{\sigma\} \times \tau \rightarrow \{\sigma \times \tau\}$. Note that $\rho_{2,(\sigma,\tau)}$, is interdefinable with the cartesian product $\bowtie_{\sigma,\tau}$: $\{\sigma\} \times \{\tau\} \rightarrow \{\sigma \times \tau\}$ by $\bowtie_{\sigma,\tau} = \mu_{\sigma\times\tau} \circ (map(\rho_{1,(\sigma,\tau)})) \circ \rho_{2,(\{\sigma\},\tau)}$ and $\rho_{2,(\sigma,\tau)} = \bowtie_{\sigma,\tau} \circ (\eta_\sigma \times id_{\{\tau\}})$.

The "*ext*" and "ext_1" constructions are derived in a standard way from the monad operations:

$$\frac{f : \sigma \rightarrow \{\tau\}}{ext(f) : \{\sigma\} \rightarrow \{\tau\}} \qquad \frac{g : \sigma \times \nu \rightarrow \{\tau\}}{ext_1(g) : \{\sigma\} \times \nu \rightarrow \{\tau\}}$$

by $ext(f) := \mu_\tau \circ (map\ f)$, $ext_1(g) := ext(g) \circ \rho_{1,(\sigma,\nu)}$. *ext*, $map(f)$, μ_σ and $\rho_{2,(\sigma,\tau)}$ can all be derived from ext_1, and can all be derived from the ext_1 construction (see also [9]).

$doubleton_\sigma : \sigma \times \sigma \rightarrow \{\sigma\}$, with the intended meaning $doubleton_\sigma(x,y) = \{x,y\}$, is defined to be $\cup_\sigma \circ (\eta_\sigma \times \eta_\sigma)$. Conversely, we could define $\cup_\sigma := \mu_\sigma \circ doubleton_{\{\sigma\}}$.

The type $\{unit\}$ plays the role of the booleans (see [9]). From the equality at base types eq_b, we can define equality at all types $eq_\sigma : \sigma \times \sigma \rightarrow \{unit\}$, intersection and difference $\cap_\sigma, -_\sigma : \{\sigma\} \times \{\sigma\} \rightarrow \{\sigma\}$, and the conditional *if then else* $: \{unit\} \times \{\sigma\} \times \{\sigma\} \rightarrow \{\sigma\}$.

Sometimes, we shall suppress the type indexes when they are clear from the context, writing $\eta, \mu, \cup, \rho_2$ etc., instead of $\eta_\sigma, \mu_\sigma, \cup_\sigma, \rho_{2,(\sigma,\tau)}$.

$\mathcal{NRA}(\Sigma, \mathcal{X})$ enjoys a certain combinatorial (functional) completeness property (see [9]):

combinatorial (functional) completeness For any function $f : \sigma \rightarrow \tau$ and any variable $x : unit \rightarrow \nu$, there exists some function $\kappa x.f : \sigma \times \nu \rightarrow \tau$, which doesn't contain the variable x, such that $\kappa x.f \circ (id, x \circ \iota_\sigma) = f$.

This property is used in [9] for the translation of the "monad calculus" into the "monad algebra". In the "monad calculus", we can write queries in a more readable form: e.g., we can define $unnest : \{\sigma \times \{\tau\}\} \rightarrow \{\sigma \times \tau\}$ as $unnest(x) := \{(u,v) \mid \exists w.(u,w) \in x \wedge v \in w\}$, instead of the almost unreadable $unnest := \mu \circ map(\rho_2)$ in the "monad algebra". We shall use freely the monad calculus style of definitions in this paper.

We interpret $\mathcal{NRA}(\Sigma, \mathcal{X})$ in Σ-models. A Σ **model** $\mathcal{B}$, consists of a family of sets $\mathcal{B}_b$ for each base type b in Σ (this gives us an **interpretation of types**, by: $[\![b]\!]_\mathcal{B} = \mathcal{B}_b$, $[\![unit]\!]_\mathcal{B} = \{()\}$, $[\![\sigma \times \tau]\!]_\mathcal{B} = [\![\sigma]\!]_\mathcal{B} \times [\![\tau]\!]_\mathcal{B}$ and $[\![\{\sigma\}]\!]_\mathcal{B} = \mathcal{P}_{fin}([\![\sigma]\!]_\mathcal{B})$ (finite sets)), and a function assigning to each function symbol $f \in \Sigma$, some function $[\![f]\!]_\mathcal{B} : [\![d_f]\!]_\mathcal{B} \rightarrow [\![c_f]\!]_\mathcal{B}$. For some **environment** - i.e. function $\rho : \mathcal{X} \rightarrow \bigcup_{(\text{All types } \sigma)} [\![\sigma]\!]_\mathcal{B}$ s.t. $\rho(x^\sigma) \in [\![\sigma]\!]_\mathcal{B}$ - we define the **interpretation of an operation** $f : \sigma \rightarrow \tau$ in $\mathcal{NRA}(\Sigma, \mathcal{X})$, to be a function $[\![f]\!]_\mathcal{B}\rho : [\![\sigma]\!]_\mathcal{B} \rightarrow [\![\tau]\!]_\mathcal{B}$, in the standard way, s.t. for each variable $x^\sigma : unit \rightarrow \sigma$, $([\![x^\sigma]\!]_\mathcal{B}\rho)(()) = \rho(x^\sigma)$.

We shall avoid writing the square brackets and the environment, $[\![\ ]\!]_\mathcal{B}\rho$, when no confusion arises.

The variables in $\mathcal{X}$ are irrelevant to the expressive power of the language, and are considered only for some technical reason. We write $\mathcal{NRA}(\Sigma)$ when $\mathcal{X} = \phi$, and simply $\mathcal{NRA}$, when, in addition, Σ has no function symbols.

The following result is from [9]:

Proposition 1 *If all functions $f \in \Sigma$ are computable in polynomial time (space), then all functions expressible in $\mathcal{NRA}(\Sigma)$ are computable in polynomial time (space).*

The proof is a straightforward induction on the structure of f.

$\mathcal{NRA}$ is equivalent to the *restricted safe calculus* or *the algebra without powerset* in [1]. It is also (almost) equivalent to the *nested algebra* in [20].

Following [15], 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 $sh(f) := max(sh(\sigma), sh(\tau))$. For $k \geq 0$, define $\mathcal{NRA}_k(\Sigma, \mathcal{X})$ to be the following sublanguage of $\mathcal{NRA}(\Sigma, \mathcal{X})$:

Types All types σ of $\mathcal{NRA}(\Sigma, \mathcal{X})$ with $sh(\sigma) \leq k$.

Operations The operations are defined in the same way as for $\mathcal{NRA}(\Sigma, \mathcal{X})$, but without exceeding the set height k. More, we replace the primitives map and μ with ext_1 in this language: then, ext is derived from ext_1[2], μ_σ is derived as $ext(id_{\{\sigma\}})$, and $map(f) = ext(\eta \circ f)$. We do this, because for defining $ext(f) = \mu \circ map(f)$, one needs a type with a higher set height, which might not be present in $\mathcal{NRA}_k(\Sigma, \mathcal{X})$; for similar reasons, we choose $\bowtie$ as a primitive, instead of ρ_2.

$\mathcal{NRA}_k(\Sigma, \mathcal{X})$ still enjoys the combinatorial (functional) completeness property (for this, it seems necessary to choose ext_1 as a primitive, instead of ext).

Note that the language $\mathcal{NRA}(\Sigma, \mathcal{X})$ is not apriori a conservative extension of $\mathcal{NRA}_k(\Sigma, \mathcal{X})$, because the former may contain operations $f : \sigma \to \tau$ constructed from expressions mentioning "intermediate" types of higher set height than k. However, the following was proven, first in [20] for the nested algebra, and later by [29] for $\mathcal{NRA}(\Sigma)$:

Theorem 1 *([29], [20]) $\forall k \geq 1$, if all function symbols in Σ have set height $\leq k$, then $\mathcal{NRA}(\Sigma)$ is a conservative extension of $\mathcal{NRA}_k(\Sigma)$*

As a consequence, the nested relational algebra "cuts down" at flat types to the relational algebra. The significance of this theorem consists in showing that, complex objects serve merely for better organizing data, not for improving the computational power of the language. Over *flat* types (i.e. products of *relations types*, which are of the form $\{b_1 \times \ldots \times b_m\}$, with b_i = either a base type, or unit), $\mathcal{NRA}(\Sigma)$ cannot express more than the *relational algebra*, as described (for example) in [22]. As a consequence, the transitive closure $tc : \{b \times b\} \to \{b \times b\}$, defined by $tc(x) = x \cup (x \circ x) \cup (x \circ x \circ x) \cup \ldots$ (here $\circ$ is relationa composition), is not expressible in $\mathcal{NRA}(\Sigma)$ (see [22], pp. 92). But the transitive closure of flat relations can be expressed in extensions of the relational algebra, for example in $DATALOG^{\neg}$ (see [22], [6]), in which all queries are still in *PTime*. This legitimates the search for extensions of $\mathcal{NRA}(\Sigma)$, which are still in polynomial time, but powerful enough to express the transitive closure.

The result in theorem 1 is in contrast with [15], who shows that the sublanguages of *CALC* become strictly more powerful, as types of larger set height are allowed; the reason is that *CALC* is, essentially, $\mathcal{NRA}(\Sigma) + powerset$, and, by allowing powersets of larger set heights as intermediate results, one can express more functions at lower types. As an example (see [1]), $tc(x)$ can be computed by first computing the *domain* of x, $dom(x) = (map(\pi_1))(x) \cup (map(\pi_2))(x)$, then computing *all* relations over $dom(x)$, namely $powerset(dom(x) \times dom(x))$, then selecting only those which are transitive and contain x, and finally taking their intersection.

[2] ext_1 is needed, instead of ext, to enable the proof of combinatorial (functional) completeness.

3 Fixpoints

Our goal is to enrich $\mathcal{NRA}$ such that it becomes at least as expressible (at flat types) as $DATALOG^{*\neg}$ (*while* queries), or $DATALOG^{\neg}$ (*fixpoint* queries): see, e.g. [6]. But simply plugging in the obvious generalization of the inflationary or noninflationary (partial) fixpoint used to extend the first order logic ([4]), gives us a language whose queries are no longer computable in polynomial time (or space). Therefore we investigate an alternative fixpoint, the *bounded fixpoint* (call it *bfix* as opposed to *fix*), due to Peter Buneman: the new constructs (inflationary and partial bounded fixpoint) are equivalent to the traditional fixpoints at flat types, but they are weak enough to keep our languages within *PTime* and *PSpace* respectively.

Define an **ordered type** to be either $\{\sigma\}$, or the product $\tau \times \nu$ of two order types τ and ν. Intuitively, an order type τ has the form $\{\sigma_1\} \times \ldots \times \{\sigma_m\}$ (but, recall, the product is *not* associative). The interpretation of an ordered type has a natural order structure, namely the component by component inclusion, under which it is a lattice with a minimal element: extending the syntax, we write $\cup, \cap : \tau \times \tau \to \tau$ and $\phi : unit \to \tau$, for the lub, glb and the minimal element of the lattice.

Consider the following fixpoint constructions:

$$\frac{f : \sigma \times \tau \to \tau}{fix\ f : \sigma \to \tau}(\tau \text{ an ordered type})$$

$$\frac{f : \sigma \times \tau \to \tau \quad g : \sigma \to \tau}{bfix(f \mid g) : \sigma \to \tau}(\tau \text{ an ordered type})$$

Two different semantics, for fixpoints, are defined in the literature: 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 $y_0 = \phi \in \tau$, $y_{n+1} = f(x, y_n)$; if there is some n for which $y_n = y_{n+1}$, then we define the **partial fixpoint** (sometimes called **noninflationary 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 $y_0 = \phi$, $y_{n+1} = y_n \cup f(x, y_n)$. The **bounded** partial (inflationary) fixpoint of f and g, is defined to be the partial (inflationary) fixpoint of $f(x, y) \cap g(x)$.

Note that the partial fixpoints $fix_p(f)$ and $bfix_p(f \mid g)$ are interpreted as *partial* functions. When Σ contains function symbols, the inflationary fixpoints may also be interpreted as partial functions: e.g., consider *nat* to be a base type, and $0 : unit \to nat$, $succ : nat \to nat$ be function symbols in Σ. Then $f : nat \times \{nat\} \to \{nat\}$, $f(x, y) := \{x\} \cup map(succ)(y)$ doesn't have an inflationary fixpoint when $nat, succ, 0$ have the standard interpretation.

We don't search for an axiomatization for the equalities valid in the 4 extensions of $\mathcal{NRA}(\Sigma, \mathcal{X})$ with fixpoints, but consider two expressions $f, g : \sigma \to \tau$ to be **equal** iff $[\![f]\!]_{\mathcal{B}}\rho = [\![g]\!]_{\mathcal{B}}\rho$, for all models $\mathcal{B}$ and environments ρ.

Proposition 2 *$\mathcal{NRA}(\Sigma, \mathcal{X}) + bfix_i$, $\mathcal{NRA}(\Sigma, \mathcal{X}) + bfix_p$, $\mathcal{NRA}(\Sigma, \mathcal{X}) + fix_i$, and $\mathcal{NRA}(\Sigma, \mathcal{X}) + fix_p$ are combinatorial (functional) complete.*

The proof simply extends the proof found in [9], to fixpoints. Note that a construction of the form:

$$\frac{f : \{\tau\} \to \{\tau\}}{fix(f) : unit \to \{\tau\}}$$

would not allow for a similar result.

Example We give 3 ways in which one can compute the powerset using the (unbounded) fixpoint. Indeed, consider $f_i : \{\sigma\} \times \{\{\sigma\}\} \to \{\{\sigma\}\}$, $i = 1, 2, 3$, given below ($ins, del : \sigma \times \{\sigma\} \to \{\sigma\}$ are: $ins(e, x) := \{e\} \cup x$, $del(e, x) := x - \{e\}$):

$$\begin{aligned}
f_1(x, Y) &= \{\phi\} \cup map(\eta)(x) \cup map(\lambda(y_1, y_2).y_1 \cup y_2)(Y \bowtie Y) \\
f_2(x, Y) &= \{\phi\} \cup map(\lambda(e, y).ins(e, y))(x \bowtie Y) \\
f_3(x, Y) &= \{x\} \cup map(\lambda(e, y).del(e, y))(x \bowtie Y)
\end{aligned}$$

Then $fix(f_i) : \{\sigma\} \to \{\{\sigma\}\}$ computes the powerset, for all $i = 1, 2, 3$, both under inflationary and partial semantics.

Example 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 $\mathcal{NRA}$), $g(x) = (\Pi_1(x) \cup \Pi_2(x)) \bowtie (\Pi_1(x) \cup \Pi_2(x))$ (where $\Pi_i : \{\sigma_1 \times \sigma_2\} \to \{\sigma_i\}$ is $map(\pi_i)$, $i = 1, 2$). Then the transitive closure of x is given by $tc(x) = bfix(f \mid g)(x)$ (both for inflationary and partial semantics). One can use tc to construct the set of all strong connected components: $scc : \{\sigma \times \sigma\} \to \{\{\sigma\}\}$ by $scc(x) := map(\lambda e.\Pi_2(\sigma_{\lambda(a,b).eq(a,e)}(tc(x))) \cap \Pi_1(\sigma_{\lambda(a,b).eq(e,b)}(tc(x))))(x)$ (where $\sigma_P : \{\tau\} \to \{\tau\}$ is the *selection*, for some predicate $P : \tau \to \{unit\}$).

Example The "same generation" problem $sg : (\sigma \times \sigma) \times \{\sigma \times \sigma\} \to \{unit\}$ asks whether two nodes a, b are at the same depth in a binary tree given by some binary relation x. For this, one computes the set of all quadruples $((a, a'), (b, b'))$, with a', b' ancestors of a and b respectively s.t. $dist(a, a') = dist(b, b')$; one stops when there is some quadruple in which either a' or b' has no direct ancestor. This set is clearly bounded by $(v \bowtie v) \bowtie (v \bowtie v)$, where $v = \Pi_1(x) \cup \Pi_2(x)$ is the set of all nodes. The "balanced tree" problem asks whether the tree given by x is balanced: simply check if there is some pair of leaves $(a, b) \in v \bowtie v$ for which $sg(a, b, x)$ is false.

For the next two propositions, we introduce the notations fix^1_p, fix^1_i, $bfix^1_p$, $bfix^1_i$ for the fixpoints restricted to only one set:

$$\frac{f : \sigma \times \{\sigma'\} \to \{\sigma'\}}{fix^1_i\ f : \sigma \to \{\sigma'\}}$$

and similarly for $bfix^1(f \mid g)$.

Proposition 3 *Under mild conditions on the functions in* Σ*, for all* $k \geq 1$*,*

$$\begin{aligned}
\mathcal{NRA}_k(\Sigma) + fix^1_p &= \mathcal{NRA}_k(\Sigma) + fix_p \\
\mathcal{NRA}_k(\Sigma) + fix^1_i &= \mathcal{NRA}_k(\Sigma) + fix_i \\
\mathcal{NRA}_k(\Sigma) + bfix^1_p &= \mathcal{NRA}_k(\Sigma) + bfix_p \\
\mathcal{NRA}_k(\Sigma) + bfix^1_i &= \mathcal{NRA}_k(\Sigma) + bfix_i
\end{aligned}$$

Proof The proof is a straightforward extension of [13], made even simpler by the presence of a boolean type (namely the type $\{unit\}$). The idea is to encode $\{\tau_1\} \times \ldots \times \{\tau_k\}$ by $\{\tau'\}$, where $\tau' = (\tau_1 \times bool) \times \ldots \times (\tau_k \times bool)$. Some $y = (y_1, \ldots, y_k) \in \{\tau_1\} \times \ldots \times \{\tau_k\}$ will be encoded by any $y' \in \{\tau'\}$, such that $\forall((z_1, b_1), \ldots, (z_k, b_k)) \in y'$ there is exactly one index i for which $b_i = true$, and:

$$\forall i, y_i = \{z_i \mid \exists z_1 \ldots \exists z_{i-1} \exists z_{i+1} \ldots \exists z_k((z_1, false), \ldots, (z_i, true), \ldots, (z_k, false)) \in y'\}.$$

So, each y_i is represented by some subset in y', which has all columns not belonging to y_i filled arbitrarily. The problem is, however, that we really have to find those arbitrary values, to put in the other columns. When $y_j \neq \phi$, then, we can simply pick the values in y_j. Else, we can still construct some arbitrary values, if we have atomic values for each base type mentioned in τ_j. So, starting with the input value $x : \sigma$, we apply repeatedly functions in Σ, trying to generate values of each base type. To keep this process finite, we impose the following condition on each $p : d_p \to c_p$ in Σ:

- For each base type b mentioned in c_p, if, for some value $z \in d_p$, $p(z)$ has some subobjects of type b, but z does not have subobjects of type b, then $p(z')$ should contain subobjects of type b for any other $z' \in d_p$.

E.g. any scalar function, like $p : b_1 \times b_2 \to b$ satisfies the condition, because $p(z)$ always returns some object of type b; $gen : nat \to \{nat\}$, $gen(n) := \{0, 1, \ldots, n-1\}$ satisfies the condition, because is always generates *numbers* from an already generated number; the same argument holds for the slightly more complicated $p : \{nat\} \to \{nat\}$, $p(x) := \{v \mid v^2 \in x\}$. An example of some functions which does *not* satisfy the above condition, is $p : \{b\} \to \{nat\}$, $p(x) = \phi$ when $card(x) \leq 10$, $p(x) = \{1\}$ when $card(x) > 10$: p has the potential of generating some number out of values of type b, but, even if we *have* values of type b, we cannot generate some number.

The rest of the proof is identical to [13], and we skip it. □

Proposition 4 *Let $\Omega = \{\Omega_\sigma\}$ be a family of function symbols indexed by the types σ, $\Omega_\sigma : unit \to \sigma$, such that, for any model $\mathcal{B}$ and environment ρ, $[\![\Omega_\sigma]\!]_{\mathcal{B}}\rho =$ the (totally) undefined function. Then*[3]*:*

$$\begin{aligned} &\mathcal{NRA}_k(\Sigma) + fix_i^1 \\ &\quad \subseteq \ \mathcal{NRA}_k(\Sigma) + fix_p \\ &\quad \subseteq \ \mathcal{NRA}_{k+1}(\Sigma \cup \Omega) + fix_i^1 \end{aligned}$$

As a consequence, 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.*

Proof (Sketch) The last inclusion is the only nontrivial one, so take some $f : \sigma \times \tau \to \tau$ in $\mathcal{NRA}_k(\Sigma) + fix_p$. Define $g : \sigma \times \{\tau\} \to \{\tau\}$ to be: $g(x, Y) = \{\phi\} \cup (map\ \lambda y.f(x, y))(Y)$ and $h(x) = (fix_i^1\ g)(x)$. Then $(fix_p\ f)(x) = if\ (\exists y \in h(x).(f(x, y) = y))\ then\ y\ else\ \Omega$ □

As in [16] (see also [12]), we get:

Proposition 5 *([16]) $\mathcal{NRA} + fix =$ 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, for each $k \geq 1$, $\mathcal{NRA}_{k+1} + fix_i$ coincides with all queries whose* time *complexity is $O(exp(k, cn))$ (for some $c > 0$), while $\mathcal{NRA}_{k+1} + fix_p$ coincides with the queries whose* space *complexity is $O(exp(k, cn))$ (for some $c > 0$).*

Proof The proof is similar to [12]. The condition $k \geq 1$ makes it possible to compute the set of all linear orders over the universe of the input at some base type b, $lo : \{b\} \to \{\{b \times b\}\}$. For $k = 0$, it is known that the two query languages obtained from fix_i and fix_p are strictly weaker than the complexity classes *PTime*

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

and *PSpace* (see [24] and [17]). The connection between the set height restriction of the language and its computational complexity is shown in [16]. $\Box$

So, by extending $\mathcal{NRA}$ with any unbounded fixpoints, we can express untractable queries, an undesired property. As an alternative, we investigate the language obtained by adding one of the two *bounded* fixpoints, $bfix_p$ or $bfix_i$. We have immediately:

Proposition 6

$$\begin{aligned} \mathcal{NRA}(\Sigma) + bfix_p &\subseteq PSpace \\ \mathcal{NRA}(\Sigma) + bfix_i &\subseteq PTime \end{aligned}$$

Proof The proof is an easy extension of the induction of proposition 5. For the fixpoint, remark that $(bfix_p\ f \mid g)(x) \subseteq g(x)$, so it stays within *PSpace*, because $g(x)$ does. For $bfix_i$, we have $(bfix_i\ f \mid g)(x) = \bigcup_{n \geq 0} y_n$ (see the definition), so there is some n, $n \leq card(g(x))$ for which $y_n = y_{n+1}$. $\Box$

Note that in contrast to Proposition 5, this result holds also for the case in which Σ contains function symbols.

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 $\mathcal{RA}(\Sigma) + bfix$ (the relational algebra over signature Σ with bounded fixpoints - to be defined).*
2. *If I is a pair-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 $\mathcal{RA}(\Sigma \cup I) + fix$.*

We first prove the theorem for the more restrictive case, when all function symbols in Σ have set height 0. Then, we show how to extend it for function symbols of set height ≤ 1 (in the presence of "set constructors" in Σ, like $gen : nat \longrightarrow \{nat\}$, $gen(n) = \{0, 1, \ldots, n-1\}$ - see [18] -, we need "index invention").

Note that, when Σ doesn't contain function symbols, then $\mathcal{NRA}_1 + bfix_p = \mathcal{NRA}_1 + fix_p$ (and similarly for $bfix_i$ and fix_i), because at flat types we can easily compute an upper bound for each fixpoint. So, we get:

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]), or to $DATALOG^{*\neg}$ ([6]).*

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

This result has as a negative consequence the fact that the inclusions from proposition 6 are strict, because the test for even cardinality is not expressible. On the other hand, no database query capable of capturing *exactly* the *PTime* queries is currently known, so $bfix_i$ and $bfix_p$ seem to offer a reasonable extension to complex objects, of the fixpoints in the relational algebra (or, equivalently, first order logic). As in the flat case, we have:

Theorem 3

$$\begin{aligned}\mathcal{NRA} + bfix_p + order &= PSpace \\ \mathcal{NRA} + bfix_i + order &= PTime\end{aligned}$$

Proof (Sketch) We follow closely [12]. Let $f : \sigma \rightarrow \tau$ be a *PTime* (or *PSpace*) *computable* query (cf. [10], i.e. invariant under isomorphisms). The bounded fixpoint is powerful enough to simulate some Turing Machine with the techniques in [12], as long as we have counters up to some polynomial of the input size. To construct such a counter, without using *powerset*, we consider every subtype σ' mentioned in σ, and collect all values of type σ' mentioned in the input x (this can be done in $\mathcal{NRA}$): call this set $v_{\sigma'} \in \{\sigma'\}$. Consider now the cartesian product of those which are nonempty: call it v. One can prove that there are constants $k, l \geq 0$ (which do not depend on x), such that $size(x) \leq 2^k card(v^l)$, where $size(x)$ is the size of the standard encoding of x (see [12]). So, if we need numbers up to $(size(x))^p$ (for $p > 0$), we can take the set $(\{false, true\}^k \times v^l)^p$, and order it by lifting the order relation at the base types (this can be done in just $\mathcal{NRA}$: see e.g. [18]). There are 2^s posibilities for the type of this set, where s is the number of subtypes σ' of σ, corresponding to all possibilities of empty sets $v_{\sigma'}$, and we choose, with a cascade of if's, the right one. Limited arithmetic function on this counter can be easily expressed, once we can express transitive closure (we need $bfix$ for that), so we can simultate a Turing Machine (again, $bfix$ suffices, since this is done a flat types, where we can compute an upper bound). Finally, observe that encoding of x, and decoding of the result, can be done with $bfix$. □

4 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 shall call our language $\mathcal{NRC}(\Sigma) + bfix$, although it is *not* derived from the monad calculus in [9]. It is strongly inspired by previous work ([1], [12], [15]), but its syntax is kept closer to the algebraic language, to make the proof of the conversions easier. Its main interest consists in the rules for range restriction, especially those connected to the fixpoint construction.

$\mathcal{NRC}$ has two syntactical categories, terms and formulas, which are defined recursively (the idea being that $\{u\ /\varphi\}$ is a term, where φ is a formula). A signature Σ is 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. All terms are typed.

terms The following are terms:

- () (the empty tuple).
- x^σ (the input variable). This variable is considered to be bound.
- $u^\sigma, v^\sigma, w^\sigma, \ldots$ (variables).
- (t_1, t_2), (the tuple of two terms t_1 and t_2).
- $f(t)$, where $f \in \Sigma$ is a function symbol and t is a term.
- $\pi_i(t)$, (the i's projection, $i = 1, 2$), where t is some term.
- $\{u^\tau\ /\varphi\}$, where u^τ is a variable and φ is some formula (the type of this term is $\{\tau\}$). u^τ becomes bound in $\{u^\tau\ /\varphi\}$ (but other variables free in φ remain free in $\{u^\tau\ /\varphi\}$).

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.
- $\exists u^\sigma \in t.\varphi$, where u^σ is a variable, t is a term (of type $\{\sigma\}$), and φ is some formula.

We impose the restriction that all input variables occurring in some term have the same type (i.e. there is only one input variable in each term).

We extend the language with a fixpoint construction:

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

We shall assume that all bounded variables in some term t or formula φ are distinct, and distinct from the free variables[4].

A **query** of type $\sigma \to \tau$ in $\mathcal{NRC}(\Sigma)$ or $\mathcal{NRC}(\Sigma) + fix$, is simply a term of type τ, having (all occurrences of) its 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 the following *unnest* query, $\{\sigma \times \{\tau\}\} \to \{\sigma \times \tau\}$: $unnest := \{u\ /\exists v \in x.\exists w \in \pi_2(v).u = (\pi_1(v), w)\}$, or in the following *nest* query, $\{\sigma \times \tau\} \to \{\sigma \times \{\tau\}\}$, $nest := \{u\ /\exists v \in x.u = (\pi_1(v), \{w\ /(\pi_1(v), w) \in x\})\}$.

Queries are interpreted in a Σ-model $\mathcal{B}$[5]. An **environment** is a mapping ρ from variables (including the input variable(s)) into $\bigcup_\sigma [\![\sigma]\!]$, preserving types. Given a model $\mathcal{B}$, an environment ρ, we interpret a term t as a (possible undefined) value $[\![t]\!]\rho : [\![\tau]\!]$, and a formula φ as a (possible undefined) truth value $[\![\varphi]\!]\rho : \{true, false\}$, in the standard way. The intuitive meaning of the formula $\exists u \in t.\varphi$, is $\exists u.(u \in t \wedge \varphi)$ (and $[\![\exists u \in t.\varphi]\!]\rho$ is defined accordingly). The interpretation of the term $\{u\ /\varphi\}$ is: $\{\alpha\ /[\![\varphi]\!]\rho[u/\alpha] = true$[6]$\}$; it is undefined, if the latter set is infinite (this will be impossible in the range restricted version of $\mathcal{NRA}$, to be defined below). When t (or φ) is a *closed* term (or formula), then the relevant part of ρ just assigns a value to the input variable x^σ in $[\![\sigma]\!]$, so $[\![t]\!]$ (or $[\![\varphi]\!]$) can be viewed as a (partial) function $[\![t]\!] : [\![\sigma]\!] \to \tau$ (or $[\![\varphi]\!] : [\![\sigma]\!] \to \{true, false\}$). For the fixpoint construction, we consider both inflationary and partial interpretations.

This language is too powerful, because it allows us to define the powerset of some set x, by $\{u^{\{\sigma\}}\ /\forall v^\sigma \in u^{\{\sigma\}}.v^\sigma \in x^{\{\sigma\}}\}$ ($\forall$ is expressed using $\exists$ and $\neg$). It also allows us to define domain dependent infinite sets, like $\{u\ /\neg u \in x\}$ [7]. Therefore, we restrict this language, by defining a *range restricted* sublanguage $(\mathcal{NRC}(\Sigma))^{rr}$, or $(\mathcal{NRC}(\Sigma) + bfix)^{rr}$. For this, we define a notion of a **range restricted** subterm t of some term or formula M ($M = \ldots t \ldots$); when such a subterm is range restricted, we underline it ($M = \ldots \underline{t} \ldots$). Also, when M is some term or formula and u some variable, we write $\underbrace{M}_{u}$ as an abbreviation for the statement "all occurrences of u in M are underlined". The following rules govern the range restriction labeling. Recall that range restriction is a property of a subterm of some term or formula M.

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

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

[5]Recall, this is a family of sets $\mathcal{B}_b$ for each base type b, together with an assignment of functions to function symbols in Σ.

[6]This implies that $[\![\varphi]\!]\rho[u/\alpha]$ is defined.

[7]The interpretation of this term is actually undefined, when the underlying domain is infinite.

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

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

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

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

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

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

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

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, even though the fixpoint variable w is *never* range restricted. This is a different approach than in [12].

We emphasize again, that range restriction is a property of *subterms* of some term or formula M. E.g. it might be possible that $\underbrace{\varphi}_{u}$, when φ is viewed as a subformula of itself, but this is never the case, when φ is viewed as a subformula of $\neg\varphi$.

We shall write $(\mathcal{NRC}(\Sigma) + fix)^{rr}$ for the **range restricted** version of $\mathcal{NRC}(\Sigma) + fix$, i.e. in which, the toplevel term is range restricted, and every bounding term of some quantifier is range restricted (i.e. the quantifiers are of the form $\exists u \in \underline{t}.\varphi$). One can easily verify that the queries *nest* and *unnest* defined before, belong to this sublanguage.

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

Proof

1. $\mathcal{NRA}(\Sigma) + bfix \subseteq (\mathcal{NRC}(\Sigma) + fix)^{rr}$. This is done straightforward, by converting any function $f : \sigma \to \tau$ from $\mathcal{NRA}(\Sigma) + bfix$, into some term t_f of type τ, with input variable of type σ. We present the most significant cases.

   $\eta : \sigma \to \{\sigma\}$ is translated to $t_\eta := \{u^\sigma \,/ u^\sigma = x^\sigma\}$.

   $\mu : \{\{\sigma\}\} \to \{\sigma\}$ is translated to $t_\mu := \{u^\sigma \,/ \exists v^{\{\sigma\}} \in x^{\{\{\sigma\}\}}.u^\sigma \in v^{\{\sigma\}}\}$. This term *is* range restricted, by the following judgement (we keep the type superscripts for the variables in this example, to make it more formal):

$$\{u^\sigma \,/\exists v^{\{\sigma\}} \in \underline{x^{\{\{\sigma\}\}}}.u^\sigma \in v^{\{\sigma\}}\}$$

$$\{u^\sigma \,/\exists v^{\{\sigma\}} \in x^{\{\{\sigma\}\}}.\underbrace{u^\sigma \in v^{\{\sigma\}}}_{v^{\{\sigma\}}}\}$$

$$\{u^\sigma \,/\exists v^{\{\sigma\}} \in x^{\{\{\sigma\}\}}.u^\sigma \in \underbrace{v^{\{\sigma\}}}_{v^{\{\sigma\}}}\}$$

$$\{u^\sigma \,/\exists v^{\{\sigma\}} \in x^{\{\{\sigma\}\}}.u^\sigma \in \underline{v^{\{\sigma\}}}\}$$

$$\{u^\sigma \,/\exists v^{\{\sigma\}} \in x^{\{\{\sigma\}\}}.\underline{u^\sigma} \in \underline{v^{\{\sigma\}}}\}$$

$$\{u^\sigma \,/\exists v^{\{\sigma\}} \in x^{\{\{\sigma\}\}}.\underbrace{u^\sigma \in v^{\{\sigma\}}}_{u^\sigma}\}$$

$$\{u^\sigma \,/\underbrace{\exists v^{\{\sigma\}} \in x^{\{\{\sigma\}\}}.u^\sigma \in v^{\{\sigma\}}}_{u^\sigma}\}$$

$$\underline{\{u^\sigma \,/\exists v^{\{\sigma\}} \in x^{\{\{\sigma\}\}}.u^\sigma \in v^{\{\sigma\}}\}}$$

   $map(f) : \{\sigma\} \to \{\tau\}$ is translated to $t_{map(f)} := \{u \,/\exists v \in x.u = t_f[x/v]\}$

   $\cup : \{\sigma\} \times \{\sigma\} \to \{\sigma\}$ is translated to $t_\cup := \{u \,/u \in \pi_1(x) \vee u \in \pi_2(x)\}$

   $not : \{unit\} \to \{unit\}$ is translated to $t_{not} := \{u \,/u = () \wedge \neg\exists u \in x.true\}$.

   $eq : b \times b \to \{unit\}$ is translated to $t_{eq} := \{u \,/u = () \wedge \pi_1(x) = \pi_2(x)\}$.

   The bounded fixpoint of $f : \sigma \times \{\tau\} \to \{\tau\}$ and $g : \sigma \to \{\tau\}$, $bfix(f \mid g) : \sigma \to \{\tau\}$, is translated into $t_{bfix(f|g)} := \mu w.\{u \,/u \in t_g \wedge u \in t_f[x/(x, w)]\}$. To prove that this term is range restricted, start with the observation that t_g is range restricted, and that x is always range restricted:

$$\mu w.\{u \,/u \in \underline{t_g} \wedge u \in t_f[x/(\underline{x}, \{v \,/v \in \underline{t_g} \wedge v \in w\})]\}$$

$$\mu w.\{u \,/u \in \underline{t_g} \wedge u \in t_f[x/(\underline{x}, \{v \,/\underline{v} \in \underline{t_g} \wedge v \in w\})]\}$$

$$\mu w.\{u \,/u \in \underline{t_g} \wedge u \in t_f[x/(\underline{x}, \{v \,/\underbrace{v \in t_g}_{v} \wedge v \in w\})]\}$$

$$\mu w.\{u \,/u \in \underline{t_g} \wedge u \in t_f[x/(\underline{x}, \{v \,/\underbrace{v \in t_g \wedge v \in w}_{v}\})]\}$$

$$\mu w.\{u \,/u \in \underline{t_g} \wedge u \in t_f[x/(\underline{x}, \underline{\{v \,/v \in t_g \wedge v \in w\}})]\}$$

$$\mu w.\{u \,/u \in \underline{t_g} \wedge u \in t_f[x/\underline{(x, \{v \,/v \in t_g \wedge v \in w\})}]\}$$

   To prove that all quantifiers are bounded by range restricted terms in $t_f[x/(x, u)]$, proceed as in the proof of $\underline{t_f}$, but replacing the axiom $\underline{x}$, with the derived fact $\underline{(x, \{v \,/v \in t_g \wedge v \in w\})}$.

2. $\mathcal{NRA}(\Sigma) + bfix \subseteq (\mathcal{NRC}(\Sigma) + fix)^{rr}$. Let σ be the type of the input variable. The translation of terms t into functions $f_t : \sigma \to \tau$, and of formulas φ into predicates $P_\varphi : \sigma \to \{unit\}$ is straightforward, with two (big) exceptions: the term constructions $t := \{u^\tau \ / \psi\}$ and $t := \mu w^{\{\tau\}}.\{u^\tau \ / \psi\}$. To build the function $f_t : \sigma \to \{\tau\}$, we first construct a function (in $\mathcal{NRA}$) $b_u : \sigma \to \{\tau\}$, which is a "bound" for u, i.e.: $[\![\psi]\!]\rho = true \Longrightarrow \rho(u) \in [\![b_u]\!](\rho(x^\sigma))$. Then, for the first term, $f_t(x^\sigma) := \sigma_{P_\psi}(b_u(x^\sigma))$, where σ_{P_ψ} is the *selection* associated to the predicate P_ψ. For the second term, (i.e. $t := \mu w^{\{\tau\}}.\{u^\tau \ / \psi\}$), let $t' := \{u^\tau \ / \psi\}$, and $t'' := t'[x^\sigma / \pi_1(x^{\sigma \times \{\tau\}}), u^\tau / \pi_2(x^{\sigma \times \{\tau\}})]$. Then, take $f_t := bfix(f_{t''}, b_u)$. We shall show how to construct the bounding function b_u by induction on the proof of the fact that t $(= \{u \ / \psi\})$ is range restricted. (In fact, when t is *not* range restricted, such a bounding function may not exists).

   Suppose ψ is a given formula. We shall consider only subterms of ψ. ψ may have free variables v^ν, some of whom are asserted to be range restricted in ψ (i.e. $\underbrace{\psi}_{v}$ is proven in a context outside of ψ). We shall assume that, for each such variable v^ν, some bounding function $c_v : \sigma \to \{\nu\}$ is provided from outside.

   First, from the proof of the range restriction of (the occurrence of) any subterm t of ψ, we show how to construct a function $b_t : \sigma \to \{\nu\}$ (where ν is the type of t) [8], satisfying the property below. To state the property, we shall write $\rho \models \underline{t}$, whenever $[\![t]\!]\rho \in [\![b_t]\!](\rho(x^\sigma))$. For v a variable assumed to be range restricted in ψ, $\rho \models \underline{v}$ means $\rho(v) \in [\![c_v]\!](\rho(x^\sigma))$. The property is:

   - If $[\![\psi]\!]\rho = true$, and $\rho \models \underline{v}$ for all variables v assumed range restricted in ψ, then $\rho \models \underline{t}$.

   Secondly, for each propositional subformula occurrence φ of ψ [9], and for each variable u^ν free in ψ, we associate to the proof of $\underbrace{\varphi}_{u}$ in ψ (under some assumptions $\underbrace{\psi}_{v}$), a function $c_u : \sigma \to \{\nu\}$[10], satisfying the following condition:

   - If $[\![\psi]\!]\rho = true$, and $[\![\varphi]\!]\rho = true$, and $\rho \models \underline{v}$ for all variables v assumed range restricted in ψ, then $\rho \models u$[11].

   We give below the functions b_t and c_u for each proof rule. Verifying the two conditions is routine.

   $\dfrac{op(\underline{t_1}, \ldots, \underline{t_m})}{\underline{op(t_1, \ldots, t_m)}}$ $\qquad b_{\underline{op(t_1,\ldots,t_m)}}(x) := map(op)(b_{\underline{t_1}}(x) \bowtie \ldots \bowtie b_{\underline{t_m}}(x))$

   $\dfrac{\underline{(t_1, t_2)}}{(\underline{t_1}, \underline{t_2})}$ $\qquad b_{\underline{t_1}}(x) := \Pi_1(b_{\underline{(t_1,t_2)}})$ etc.

   $\overline{\underline{x}}$(x is the input variable) $\qquad b_x(x) := \{x\}$

   $\dfrac{t_1 \in \underline{t_2}}{\underline{t_1} \in \underline{t_2}}$ $\qquad b_{\underline{t_1}}(x) := \mu(b_{\underline{t_2}}(x))$ $\qquad \dfrac{\underline{t_1} = t_2}{\underline{t_1} = \underline{t_2}}$ $\qquad b_{\underline{t_2}} := b_{\underline{t_2}}$

   $\underbrace{\dfrac{\underline{u}}{u}}_{u}$ $\qquad c_u := b_u$

[8] t in b_t is an *occurrence* in ψ, not just a term: different b_t's may arise for different occurrences of the same term. More, even for some occurrence t, b_t is not necessarily unique, because the proof of $\underline{t}$ is not necessarily unique.

[9] I.e. φ is either ψ, or $\psi = \psi_1 \ op \ \psi_2$, with $op = \vee, \wedge$, and φ is a propositional subformula of ψ_1 or ψ_2.

[10] The notation here is even more ambiguous, because c_u, in fact, depends on ψ, the occurrence φ, the variable u, all the functions c_v, and the proof of $\underbrace{\varphi}_{u}$.

[11] I.e. $\rho(u) \in [\![c_u]\!](\rho(x))$.

$$\frac{\underbrace{u}_{u}}{\underline{u}} \qquad b_u := c_u$$

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

$$\frac{op(\underbrace{t_1}_{u},\ldots,\underbrace{t_m}_{u})}{\underbrace{op(t_1,\ldots,t_m)}_{u}} \qquad \text{the intersection of the } c_u\text{'s above}$$

$$\frac{}{\underbrace{false}_{u}} \qquad c_u(x) := \phi$$

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

take the c_u from the premise

$$\frac{\{u\ /\underbrace{\varphi}_{u}\}}{\underline{\{u\ /\varphi\}}} \qquad b_{\{u\ /\varphi\}} := c_u$$

$$\frac{\mu w.\{u\ /\underbrace{\varphi}_{u}\}}{\underline{\mu w.\{u\ /\varphi\}}} \qquad b_{\mu w.\{u\ /\varphi\}} := c_u$$

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

Here, add $c_u := b_{\underline{t}}$ to the premises of φ (see the next two rules).

$$\frac{\underbrace{\exists u \in t.\varphi}_{v}}{\exists u \in t.\underbrace{\varphi}_{v}} \qquad \text{add } c_v \text{ to the premises of } \varphi$$

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

Using the additional premises from the preceding two rules, in the proofsystem associated to the formula φ, we get some c_v (associated to φ). Take c_v associated to ψ, to be the same.

$$\dfrac{\underbrace{\varphi}_{u} \wedge \psi}{\underbrace{\varphi \wedge \psi}_{u}} \qquad \dfrac{\varphi \wedge \underbrace{\psi}_{u}}{\underbrace{\varphi \wedge \psi}_{u}} \qquad \dfrac{\underbrace{\varphi \wedge \psi}_{u}}{\underbrace{\varphi}_{u} \wedge \underbrace{\psi}_{u}} \qquad \text{same } c_v$$

$$\dfrac{\underbrace{\varphi}_{u} \vee \underbrace{\psi}_{u}}{\underbrace{\varphi \vee \psi}_{u}} \qquad c_u(x) := c'_u(x) \cup c''_u(x)$$

$$\dfrac{\underbrace{\varphi \vee \psi}_{u}}{\underbrace{\varphi}_{u} \vee \underbrace{\psi}_{u}} \qquad \text{same } c_u$$

□

5 Encoding of finite functions in the relational algebra

A **scalar type** is either a base type b, or *unit*, or a product of scalar types. A **flat type** is either $\{t\}$, with t some scalar type, or $\sigma \times \tau$, with σ, τ flat types. Note that a flat type is an ordered type.

We inted to define the relational algebra over some signature Σ, $\mathcal{RA}(\Sigma)$, to be a collection of operations over flat types, to satisfy the following:

1. It should coincide with the "traditional" relational algebra, when Σ has no function symbols (cf., e.g., [22]), and it should be a "reasonable" extension otherwise.
2. It should be inductively generated.
3. $\mathcal{NRA}_1(\Sigma)$ should be a conservative extension of $\mathcal{RA}(\Sigma)$.

By simply defining $\mathcal{RA}(\Sigma)$ to be the set of all functions $f : \sigma \to \tau$ in $\mathcal{NRA}_1(\Sigma)$ over flat types, we violate both 1 and 2. Indeed, consider some nonmonotone function $f : \{s\} \to \{t\}$ in the "traditional" relational algebra: it is not obvious that $ext(f \circ \eta)$ is also a relational algebra expression, but it *does* belong to $\mathcal{NRA}_1(\Sigma)$! Induction is also prohibited by the fact that $g \circ f$ may be in $\mathcal{RA}(\Sigma)$, although neither f nor g is in $\mathcal{RA}(\Sigma)$ (when the intermediate type is not a flat type).

To fulfill these desiderates, and to keep our constructions simple, we restrict ourselves, for the beginning, to signatures Σ containing only function symbols of set height 0.

So, we define **the relational algebra** $\mathcal{RA}(\Sigma, \mathcal{X})$ to be the following sublanguage of $\mathcal{NRA}_1(\Sigma, \mathcal{X})$: its types are the flat types, and its operations are closed under composition and pairing, and contain the identities, the projections, and the following:

union, difference $\cup, - : \{t\} \times \{t\} \to \{t\}$.

the empty set $\phi : \sigma \to \tau$, for all flat types σ, τ.

cartesian product $\bowtie: \{t_1\} \times \{t_2\} \rightarrow \{t_1 \times t_2\}$.

projections and scalar functions $map(f) : \{t\} \rightarrow \{t'\}$ for each $f : t \rightarrow t'$ in $\mathcal{NRA}_0(\Sigma)$.

selections $\sigma_P : \{t\} \rightarrow \{t\}$, for every $P : t \rightarrow \{unit\}$ in $\mathcal{NRA}_1(\Sigma)$. σ_P is defined to be $ext(\bar{P})$, where $\bar{P}$ is $t \xrightarrow{P\times\eta} \{unit\} \times \{t\} \xrightarrow{\bowtie} \{unit \times t\} \xrightarrow{map(\pi_2)} \{t\}$. Note that P cannot contain variables.

nonstrict operations $not : \{unit\} \rightarrow \{unit\}$ and $x^{\sigma,\tau} : \sigma \rightarrow \tau$, for each variable $x^\tau \in \mathcal{X}$, and flat types σ, τ. The latter is defined to be $\sigma \xrightarrow{\iota_\sigma} unit \xrightarrow{x^\tau} \tau$.

We consider the extensions $\mathcal{RA}(\Sigma,\mathcal{X}) + bfix$ and $\mathcal{RA}(\Sigma,\mathcal{X}) + fix$, by closing $\mathcal{RA}(\Sigma,\mathcal{X})$ under the bounded or the unbounded fixpoint constructions. Note that bounding the fixpoints is *not* redundant when Σ contains some function symbols.

When $\pi_i : t_1 \times t_2 \rightarrow t_i$ is a projection, then $map(\pi_i) : \{t_1 \times t_2\} \rightarrow \{t_i\}$ is the "relational algebra projection", denoted with Π_i. Consider now some function $f : t_1 \rightarrow t_2$ in Σ, and let s be some scalar type. Then $map(f \times id) : \{t_1 \times s\} \rightarrow \{t_2 \times s\}$ simply applies f to each element in the first column(s) of its input relation. This suggests that the $map(f)$ expressions are either already in the "traditional" relational algebra (the "database projections"), or they form some reasonable extension of it, when Σ contains function symbols.

Also, $\sigma_{eq\circ\pi_1} : \{(t \times t) \times s\} \rightarrow \{(t \times t) \times s\}$ simply selects those elements whose first and second component coincide (denoted with $\sigma_{1=2}$ in [22]). This suggests that the σ_P expressions are not more powerful than ordinary selections.

Proposition 7 *$\mathcal{RA}(\Sigma,\mathcal{X})$ (+bfix or +fix) is combinatorial (functional) complete, in the following sense:*

combinatorial (functional) completeness *For any function $f : \sigma \rightarrow \tau$ in $\mathcal{RA}(\Sigma,\mathcal{X})$ (+bfix or +fix), and any variable $x^{\xi,\zeta} : \xi \rightarrow \zeta$, there is some function $\kappa x.f : \sigma \times \zeta \rightarrow \tau$ such that $f = \kappa x.f \circ (id, \phi)$.*

The proof is trivial, but only because predicates $P : t \rightarrow \{unit\}$, occurring in some selection $\sigma_P : \{t\} \rightarrow \{t\}$, are not allowed to contain variables.

Our final goal is to translate $\mathcal{NRA}(\Sigma,\mathcal{X})$ (+$bfix$ or +fix) into $\mathcal{RA}(\Sigma,\mathcal{X})$ (+$bfix$ or +fix). The hard part is to simulate the set construction $\{\_\}$ in $\mathcal{RA}(\Sigma,\mathcal{X})$. For this, we represent an element of $\{\sigma\}$, as the range of some partial, finite function $\psi : t \Rightarrow \sigma$, where t is some scalar type. Not surprisingly, such functions *can* be encoded in $\mathcal{RA}$. To show that, we start by encoding total functions $\varphi : t \rightarrow \sigma$ with **finite support** (i.e. for which $supp(\varphi) = \{x \in t \;/ \varphi(x) \neq \phi\}$ is finite):

Definition 1 *Let t be a scalar type (in $\mathcal{NRA}_0$) and σ be a flat type. Define the flat type $[t \rightarrow \sigma]$, and $ap_\sigma : [t \rightarrow \sigma] \times t \rightarrow \sigma$, by induction on the structure of σ:*

1. *$[t \rightarrow \{t'\}] := \{t \times t'\}$, and $ap_{t'} := \{t \times t'\} \times t \xrightarrow{\rho_1} \{(t \times t') \times t\} \xrightarrow{\sigma_P} \{t'\}$, where $P := (t \times t') \times t \xrightarrow{eq\circ(\pi_1\circ\pi_1,\pi_2)} \{unit\}$.*
2. *$[t \rightarrow \sigma \times \tau] := [t \rightarrow \sigma] \times [t \rightarrow \tau]$, and $ap_{\sigma\times\tau} := (ap_\sigma \times ap_\tau) \circ ((\pi_1 \circ \pi_1, \pi_2), (\pi_2 \circ \pi_1, \pi_2))$.*

When $\varphi : t \rightarrow \sigma$ has a finite support, and $f : \sigma \rightarrow \tau$ is **strict** (i.e. $f(\phi) = \phi$), then $f \circ \varphi : t \rightarrow \tau$ still has a finite support. This justifies the following:

Proposition 8 *Let $\mathcal{RA}_\perp(\Sigma)$ (+$bfix$ or +fix) be the* **strict** *fragment of $\mathcal{RA}(\Sigma,\mathcal{X})$ (+$bfix$ or +fix) (i.e. without not and variables). Then for each $f:\sigma\to\tau$ in $\mathcal{RA}_\perp(\Sigma)$ (+$bfix$ or +fix), there exists $[t\to f]:[t\to\sigma]\to[t\to\tau]$ in $\mathcal{RA}_\perp(\Sigma)$, such that the following diagram commutes:*

$$\begin{tikzcd}
{[t\to\sigma]\times t} \arrow[r, "{[t\to f]\times id}"] \arrow[d, "ap_\sigma"] & {[t\to\tau]\times t} \arrow[d, "ap_\tau"] \\
\sigma \arrow[r, "f"] & \tau
\end{tikzcd}$$

Intuitively, $[t\to f](\varphi)=f\circ\varphi$.

Proof By straightforward induction on the structure of f: $[t\to id]:=id$, $[t\to g\circ f]:=[t\to g]\circ[t\to f]$, $[t\to\cup]:=\cup$, $[t\to -]:=-$, $[t\to\phi]:=\phi$, $[t\to\bowtie]:=\{t\times t_1\}\times\{t\times t_2\}\xrightarrow{\bowtie}\{(t\times t_1)\times(t\times t_2)\}\xrightarrow{\sigma_P}\{(t\times t_1)\times(t\times t_2)\}\xrightarrow{map(\pi_1\circ\pi_1,(\pi_2\circ\pi_1,\pi_2\circ\pi_2))}\{t\times(t_1\times t_2)\}$, where $P:=(t\times t_1)\times(t\times t_2)\xrightarrow{eq\circ(\pi_1\circ\pi_1,\pi_1\circ\pi_2)}\{unit\}$, $[t\to\sigma_P]:=\sigma_{P\circ\pi_2}$, $[t\to bfix(f\mid g)]:=bfix([t\to f]\mid[t\to g])$, $[t\to fix(f)]:=fix[t\to f]$. □

Note that $[t_1\to[t_2\to\sigma]]\sim[t_1\times t_2\to\sigma]$ and $[unit\to\sigma]\sim\sigma$. For $f:t_1\to t_2$, define $[f\to\sigma]^{-1}:[t_1\to\sigma]\to[t_2\to\sigma]$ to be: $[f\to\{t\}]^{-1}:=map(f\times id)$, $[f\to(\sigma\times\tau)]^{-1}:=[f\to\sigma]^{-1}\times[f\to\tau]^{-1}$. We have $[f\to\sigma]^{-1}(\varphi\circ f)=f$, so, when f is injective, $[f\to\sigma]$ is indeed the inverse of the composition to the right, as suggested by its notation.

Now we proceed to encode partial, finite, functions $\psi:t\Rightarrow\{\sigma\}$ as relation pairs $rel(\psi)=(d,\varphi)\in\{t\}\times[t\to\sigma]$, where d is the domain of ψ and φ is the function with a finite support defined by $\varphi(x)=\psi(x)$ when $\psi(x)$ is defined, and $\varphi(x)=\phi$ otherwise. Define the flat type $[t\Rightarrow\sigma]:=\{t\}\times[t\to\sigma]$. Note that only those elements (d,φ) from $[t\Rightarrow\sigma]$ which satisfy $supp(\varphi)\subseteq d$ are *valid* encodings, in the above sense, of some partial, finite function ψ. The function $decode_\sigma:[t\Rightarrow\sigma]\to\sigma$, which computes the range of some partial function, is $decode_\sigma(d,\varphi):=[t\Rightarrow\sigma]\xrightarrow{\pi_2}[t\to\sigma]\xrightarrow{[\iota_t\to\sigma]^{-1}}[unit\to\sigma]\sim\sigma$. We also define $Ap_\sigma:[t\Rightarrow\sigma]\times t\to\sigma$ to be (informally): $Ap_\sigma((d,\varphi),x)=if\ x\in d\ then\ ap_\sigma(\varphi,x)\ else\ \Omega_\sigma$. Ap_σ is in $\mathcal{RA}(\Omega)$ (recall that Ω_σ is the totally undefined term of type σ). We have:

Proposition 9 (The Map Lemma) *For each $f:\sigma\to\tau$ in $\mathcal{RA}(\Sigma,\mathcal{X})$, there is some $[t\Rightarrow f]:[t\Rightarrow\sigma]\to[t\Rightarrow\tau]$ in $\mathcal{RA}(\Sigma,\mathcal{X})$, such that the following diagram commutes:*

$$\begin{tikzcd}
{[t \Rightarrow \sigma] \times t} \arrow[r, "{[t \Rightarrow f] \times id}"] \arrow[d, "Ap_\sigma"] & {[t \Rightarrow \tau] \times t} \arrow[d, "Ap_\tau"] \\
\sigma \arrow[r, "f"] & \tau
\end{tikzcd}$$

Proof The induction proceeds as in prop 8 for the strict operations. For the nonstrict operations, take $[t \Rightarrow not](d, \varphi) := (d, (d \bowtie \{()\}) - \varphi)$ and $[t \Rightarrow x](d, \varphi) := (d, d \bowtie x)$. □

Given some $f : \sigma \times \nu \to \tau$, we can take advantage of the presence of the variables in our language, to define $[t \Rightarrow_1 f] : [t \Rightarrow \sigma] \times \nu \to [t \Rightarrow \tau]$: choose some variable $x^{\sigma,\nu} : \sigma \to \nu$, not occurring in f, and apply the map lemma to $g : \sigma \xrightarrow{(d, x^{\sigma,\nu})} \sigma \times \nu \xrightarrow{f} \tau$. Then, form $[t \Rightarrow g] : [t \Rightarrow \sigma] \to [t \Rightarrow \tau]$, and define $[t \Rightarrow_1 f] := \kappa x.[t \Rightarrow g]$.

The Map Lemma implies that, whenever $f : \{t\} \to \{t'\}$ is in $\mathcal{RA}(\Sigma)$, then $ext(f \circ \eta) : \{t\} \to \{t'\}$ is also in $\mathcal{RA}(\Sigma)$[12]. Indeed, $ext(f \circ \eta) := \{t\} \xrightarrow{(id, ID)} [t \Rightarrow \{t\}] \xrightarrow{[t \Rightarrow f]} [t \Rightarrow \{t'\}] \xrightarrow{decode_{\{t'\}}} \{t'\}$, where $ID(x) = \{(z, z) \ / z \in x\}$.

Now we are ready to define the encoding of $\mathcal{NRA}$ into flat relations.

6 Encoding $\mathcal{NRA}$ ($+bfix$ or $+fix$) into flat relations with pair-indexes

A **pair index** is some base type I equipped with a binary function $pair : I \times I \to I$ and two constants $left, right : unit \to I$. A model for I is a **pair-index structure** $(\mathcal{I}, Pair, Left, Right)$, where $Pair : \mathcal{I} \times \mathcal{I} \to \mathcal{I}$ is injective (this implies that $\mathcal{I}$ is infinite), and $Left, Right \in \mathcal{I}$ are distinct. We abbreviate $\Sigma \cup I$ for $\Sigma \cup \{I, pair, left, right\}$. If $\mathcal{B}$ is a Σ model and $\mathcal{I}$ is a pair-index structure, we write $\mathcal{B} \cup \mathcal{I}$ for the $\Sigma \cup I$ model obtained from $\mathcal{B}$ by joining $\mathcal{I}$.

I plays the role of **index set**, used in [23] to encode the nested relational algebra into the relational algebra. (The usefulness of using indexes in the Nested Relational Algebra was also recognized in [25].) In contrast with [23], we don't need inventions, but use *pair*, *left* and *right* instead. *pair* is motivated by the necessity to encode the "flatten" operation $\mu : \{\{\sigma\}\} \to \{\sigma\}$. Although the encoding in [23] is easier to understand on particular examples, we feel that the encoding presented here is formally better motivated, and enables us to extend the conservativity result to bounded fixpoints.

The translation of $\mathcal{NRA}(\Sigma) + bfix$ (or $+fix$) into $\mathcal{RA}(\Sigma \cup I) + bfix$ ($+fix$) is given by:

[12]This is not intended to be a proof of $\mathcal{NRA}_1(\Sigma)$ being a conservative extension of $\mathcal{RA}(\Sigma)$

The translation of types $\sigma \leadsto \pi_\sigma$ is: $\pi_b := \{b\}$ for any base type b, $\pi_{unit} := \{unit\}$, $\pi_{\sigma\times\tau} := \pi_\sigma \times \pi_\tau$, $\pi_{\{\sigma\}} := [I \Rightarrow \pi_\sigma]$.

The encoding relations $\sim_\sigma \subseteq [\![\sigma]\!]_{\mathcal{B}} \times [\![\pi_\sigma]\!]_{\mathcal{B}\cup\mathcal{I}}$, defined by:

$$\frac{}{x \sim_b \{x\}} \qquad \frac{}{() \sim_{unit} \{()\}} \qquad \frac{x \sim_\sigma r \quad y \sim_\sigma q}{(x,y) \sim_{\sigma\times\tau} (r,q)}$$

$$\frac{\omega : I \Rightarrow \sigma \quad \psi : I \Rightarrow \pi_\sigma \quad \omega \sim_\sigma \psi}{range(\omega) \sim_{\{\sigma\}} rel(\psi)}$$

For the last rule, ω and ψ are partial, finite functions, and $\omega \sim_\sigma \psi$ means that $\forall i \in I$, $\omega(i)$ and $\psi(i)$ are either both undefined, or $\omega(i) \sim_\sigma \psi(i)$. Recall that $rel(\psi) \in [I \Rightarrow \pi_\sigma]$ is the encoding of the partial, finite function ψ.

$x \sim_\sigma r$ means that x *may* be encoded by r. It is *total* (*any* x is encoded by *some* r), *not* functional (one might use different indexes to encode the same set), and *not* surjective (for two reasons: only sets of cardinality 1 in π_b and π_{unit} are encodings of *something*, and only elements of the form $rel(\psi)$ in $\pi_{\{\sigma\}}$ are valid encodings). However, the encoding is injective: $x \sim_\sigma r$ and $x \sim_\sigma r'$ implies $r = r'$.

The encoding of operations $(f : \sigma \to \tau) \leadsto (R_f : \pi_\sigma \to \pi_\tau)$ (to be described below) is such that $f \sim R_f$, i.e. for any models $\mathcal{B}$ and $\mathcal{B}\cup\mathcal{I}$, the following **soundness property** holds:

- $\forall x, r$, $x \sim_\sigma r$ implies $[\![f]\!]_{\mathcal{B}}(x) \sim_\tau [\![R_f]\!]_{\mathcal{B}\cup\mathcal{I}}(r)$.

The encoding of the operations $f \leadsto R_f$ proceeds in two steps. First, one encodes all operations f from $\mathcal{NRA}(\Sigma)$ (i.e. without fixpoints). For μ, we use *pair* (and rely on the injectiveness of *Pair* when proving soundness), for *doubleton* : $\sigma \times \sigma \to \{\sigma\}$ we use *left* and *right* and rely on *Left* $\neq$ *Right* ($\cup$ is defined indirectly, using *doubleton*), $R_{map(f)} := [I \Rightarrow R_f]$ (here we use the Map Lemma), and 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 $R_f : \{t_1\} \times \ldots \times \{t_m\} \to \{t\}$, $R_f(r_1, \ldots, r_m) := map(f)(r_1 \bowtie \ldots \bowtie r_m)$. 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 an ordered 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, because r and q might accidentally use the same index for encoding different elements. Therefore, we need the following two lemmas, to encode $bfix(f \mid g)$. However, *it is* the case that $x \sim_\sigma r$ implies $x = \phi \Leftrightarrow r = \phi$, for all ordered types σ.

Lemma 1 (Bounded additiveness of $\sim$) *Let σ be some ordered type and $x_1, x_2, x_3 \in \sigma$, $r_1, r_2, r_3 \in \pi_\sigma$ such that $x_i \sim_\sigma r_i$, $i = 1,2,3$ and $x_1 \subseteq x_3, x_2 \subseteq x_3$, $r_1 \subseteq r_3, r_2 \subseteq r_3$. Then $x_1 \cup x_2 \sim_\sigma r_1 \cup r_2$.*

Lemma 2 (Improved intersection) *For each ordered type σ there is some function $intersect_\sigma : \pi_\sigma \times \pi_\sigma \to \pi_\sigma$ in $\mathcal{RA}$, such that $x \sim_\sigma r$ and $y \sim_\sigma q$ implies $x \cap y \sim_\sigma intersect_\sigma(r,q)$ and $intersect_\sigma(r,q) \subseteq q$.*

intersect simply selects those elements from q which occur in r. It is not necessarily identical to $R_\cap$ (we didn't specify the way $R_\cap$ was derived), but it does a similar job. The $[t \Rightarrow_1 \_]$ construction, described after the Map Lemma, is used here. The translation of the equality at type τ, $eq_\tau : \tau \times \tau \to \{unit\}$, R_{eq_τ}, is used here in an essential way. When τ is not a base type, eq_τ is a derived operation (and its translation is a complicated function in $\mathcal{RA}$), so we need to make the full translation of $\mathcal{NRA}(\Sigma)$ into $\mathcal{RA}(\Sigma)$ before proving this lemma.

Using these two lemmas, the translation of $bfix(f \mid g) : \sigma \to \tau$ (τ an ordered type) is: $R_{bfix(f|g)} := bfix(Q \mid R_g)$ where, informally, $Q(r,q) := intersect_\tau(R_f(r,q), R_g(r))$. Then $y_0(= \phi) \sim_\tau q_0(= \phi)$ and $x \sim_\sigma r, y_n \sim_\tau q_n$

implies both $f(x, y_n) \cap g(x) \sim_\tau Q(r, q_n) \cap R_g(r)$ and $y_n \cup f(x, y_n) \cap g(x) \sim_\tau q_n \cup Q(r, q_n) \cap R_g(r)$; this concludes both $bfix_p(f, g) \sim bfix_p(Q, R_g)$ and $bfix_i(f, g) \sim bfix_i(Q, R_g)$.

In the following, we need to to consider a restricted class of pair-index structures $(\mathcal{I}, Pair, Left, Right)$. Call $i \in \mathcal{I}$ an **atom**, if $\forall x, y \in \mathcal{I}$, $i \neq Pair(x, y)$. Call $\mathcal{I}$ and **atomic pair-index structure**, if *Left*, *Right* are atoms, $\mathcal{I}$ has infinitely many atoms, and is generated by them. This implies some acyclicity of *Pair*, e.g. $Pair(x, Pair(y, z)) \neq y$ etc. From now on, we implicitly assume all pair-index structures to be atomic.

Lemma 3 *Let σ be an ordered type. Then there is an $\mathcal{RA} + fix$ expression $rename : \pi_\sigma \times \pi_\sigma \to \pi_\sigma$ such that, for all $x \sim_\sigma r$, $y \sim_\sigma q$ and $x \cap y = \phi$, we have $x \sim_\sigma rename(r, q)$ and $x \cup y \sim rename(r, q) \cup q$.*

rename has to repeatedly rename the indexes in r, e.g. by replacing every i with $pair(left, i)$, until a set of indexes is reached, which is disjoint from the indexes in q. The acyclicity condition assures that this process will terminate. Then the union with q is indeed an encoding of $x \cup y$. An unbounded fixpoint is necessary.

To translate the unbounded fixpoint $fixf$, take $R_{fixf} := fix\ Q$, where $Q(r, q) := let\ q' = R_f(r, q),\ q'' = intersect(q', q)\ in\ rename(R_-(q', q), q'') \cup q''$ (R_- is the translation of $-$). This complicated expression is needed to insure that $y_{n+1} = y_n$ implies $q_{n+1} = q_n$.

This completes the proof of the conservativity result for the unbounded fixpoint:

Theorem 5 *If I is a pair index, then $\mathcal{NRA}(\Sigma \cup I)$ is a conservative extension of $\mathcal{RA}(\Sigma \cup I) + fix$.*

7 Tagged indexes

The function $pair : I \times I \to I$ and the two constants $left, right$ in I, together with some semantic constraints (injectivity of *Pair* and atomicity), proved to be enough to encode $\mathcal{NRA}$ into $\mathcal{RA}$. We don't try to express a general encoding function $\sigma \to \pi_\sigma$ in $\mathcal{NRA}$ - this would inevitably lead to the need of index invention, an approach taken in [23]. However, the decoding function $\pi_\sigma \to \sigma$ is expressible in $\mathcal{NRA}$, and it is even in $\mathcal{RA}$, when σ is flat.

For the conservativity result, we start with some function $f : \sigma \to \tau$ in $\mathcal{NRA}(\Sigma) + bfix$, and consider first $R_f : \pi_\sigma \to \pi_\tau$. Compose it with the decoding function $decode_\tau : \pi_\tau \to \tau$ to get $Q_f : \pi_\sigma \to \tau$ in $\mathcal{RA}(\Sigma \cup I) + bfix$, with the property that in all models $\mathcal{B} \cup \mathcal{I}$, $x \sim_\sigma r$ implies $f(x) = Q_f(r)$.

The we perform two translations on Q_f: $Q_f \leadsto \bar{Q}_f$ and $\bar{Q}_f \leadsto \tilde{\bar{Q}}_f$. $\bar{Q}_f$ is in the language $\mathcal{RA}(\Sigma \cup \bar{I}) + bfix$ (where $\bar{I}$ is a new signature, to be described), while $\tilde{\bar{Q}}_f$ is in $\mathcal{RA}(\Sigma) + bfix$. Combining the, rather subtle, relationships between the interpretations of $Q_f, \bar{Q}_f$ and $\tilde{\bar{Q}}_f$ (in 3 different kind of models !), we get the desired result.

Suppose $\sigma = \{t_1\} \times \ldots \times \{t_m\}$ [13], and that the input to f is $x = (x_1, \ldots, x_m)$. Consider a set $T_0 = \{\tau_1, \ldots, \tau_m\}$ of **basic tags**, and define **a tag** to be either $\tau_k \in T_0$, or $LEFT$ or $RIGHT$ or $< t, t' >$ where t, t' are tags. Let T stand for the set of tags. Examples of tags are: $< \tau_2, < LEFT, \tau_5 >>$, $<< RIGHT, RIGHT >, < \tau_1, \tau_1 >>$ etc.

The signature $\bar{I}$ is defined to contain a type $\bar{I}_t$ for each tag t, and function symbols $\overline{left} : unit \to \bar{I}_{LEFT}$, $\overline{right} : unit \to \bar{I}_{RIGHT}$ and $\overline{pair}_{t,t'} : \bar{I}_t \times \bar{I}'_t \to \bar{I}_{<t,t'>}$, for all $t, t' \in T$. A model for $\bar{I}$ is $(\{\bar{\mathcal{I}}\}_{t \in T}, \overline{Pair}_{t,t'}, \overline{Left}, \overline{Right})$, where we always assume $\overline{Pair}_{<t,t'>} : \bar{\mathcal{I}}_t \times \bar{\mathcal{I}}_{t'} \to \bar{\mathcal{I}}_{<t,t'>}$ to be **bijections**, and $\bar{\mathcal{I}}_{LEFT} = \{\overline{Left}\}$, $\bar{\mathcal{I}}_{RIGHT} = \{\overline{Right}\}$. We call such a structure a **tagged pair-index structure**, and

[13] The cartesian product is not associative in our language, so consider that some parentheses are present, in some arbitrary order.

abbreviate it with $\bar{\mathcal{I}}$. Sometimes we simply call $\mathcal{RA}(\Sigma \cup \bar{I})$ and $\mathcal{RA}(\Sigma \cup I)$ the **tagged** and the **untagged** language.

We say that a tagged pair-index structure $\bar{\mathcal{I}}$ **is embedded** into some pair-index structure $\mathcal{I}$, $\bar{\mathcal{I}} \sqsubseteq \mathcal{I}$, if $\forall t \in T, \bar{\mathcal{I}}_t \subseteq \mathcal{I}$, $\overline{Pair_{<t,t'>}}$ is the restriction of $Pair$, $\overline{Left} = Left$, $\overline{Right} = Right$, and, for all basic tags $\tau_1, \ldots, \tau_m$, $\bar{\mathcal{I}}_{\tau_1}, \ldots, \bar{\mathcal{I}}_{\tau_m}$ are disjoint sets of atoms, which do not contain $Left$ or $Right$.

There is an obvious translation from the tagged to the untagged language, $erase : (\mathcal{RA}(\Sigma \cup \bar{I}) + bfix) \rightarrow (\mathcal{RA}(\Sigma \cup I) + bfix)$ which simply erases the tags of the indexes. When $\bar{\mathcal{I}} \sqsubseteq \mathcal{I}$, then $[\![\bar{\sigma}]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}} \subseteq [\![erase(\bar{\sigma})]\!]_{\mathcal{B}\cup\mathcal{I}}$, for all types $\bar{\sigma}$ in the tagged language, and one can easily check that, for every $\bar{f} : \bar{\sigma} \rightarrow \bar{\tau}$, $\forall x \in [\![\bar{\sigma}]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}, [\![\bar{f}]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}(x) = [\![erase(\bar{f})]\!]_{\mathcal{B}\cup\mathcal{I}}(x)$. The following lemma shows that we can enforce tags on any untagged function f.

Lemma 4 (Tagging lemma) *Let $f : \sigma \rightarrow \tau$ be a function in the untagged language, and $\bar{\sigma}$ be such that $erase(\bar{\sigma}) = \sigma$. Then, there are some q tagged types $\bar{\tau}_1, \ldots \bar{\tau}_q$, and q tagged functions $\bar{f}_k : \bar{\sigma} \rightarrow \bar{\tau}_k$, such that $erase(\bar{\tau}_1) = \ldots = erase(\,_q) = \tau$, and for all $\bar{\mathcal{I}} \sqsubseteq \mathcal{I}$,$\bar{\tau}$we have $[\![f]\!]_{\mathcal{B}\cup\mathcal{I}}(x) = [\![\bar{f}_1]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}(x) \cup \ldots \cup [\![\bar{f}_q]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}(x)$, where $x \in [\![\sigma]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}$.*

Proof (Sketch) The tagging strategy of f is guided by the observation that, whenever $\bar{\mathcal{I}} \sqsubseteq \mathcal{I}$, we have $\bar{\mathcal{I}}_t \cap \bar{\mathcal{I}}_{t'} = \phi$, for $t \neq t'$. We tag f by induction on the structure of f. The most relevant cases are $f = -$ and $f = \cup$. For $f = -, f : \{t\} \times \{t\} \rightarrow \{t\}$, we are given two tagged types $\bar{t}_1$ and $\bar{t}_2$, such that $erase(\bar{t}_1) = erase(\bar{t}_2) = t$. If $\bar{t}_1 = \bar{t}_2$, then take $q = 1$, $\bar{\tau}_1 = \{\bar{t}_1\}$ and $\bar{f}_1 = -$. But when $\bar{t}_1 \neq \bar{t}_2$, then take $q = 1$, $\bar{\tau}_1 = \{\bar{t}_1\}$ and $\bar{f}_1 = \pi_1$ (the first projection): indeed, $[\![\bar{t}_1]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}$ and $[\![\bar{t}_2]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}$ are disjoint, so the difference equals always the first argument. The case $f = \cup$ forces us to values $q > 1$: simply take $q = 2$, $\bar{\tau}_1 = \{\bar{t}_1\}$, $\bar{\tau}_2 = \{\bar{t}_2\}$, and $\bar{f}_1 = \pi_1, \bar{f}_2 = \pi_2$.

We also illustrate the case $f = map(g)$, for, say, $g : (I \times I) \times \nu \rightarrow I \times \nu$, $g = pair \times id$. Then $\bar{\sigma} = (\bar{I}_t \times \bar{I}_{t'}) \times \bar{\nu}$, for some tags t, t': here we take $\bar{\tau} := \bar{I}_{<t,t'>} \times \nu$ and $\overline{map(pair \times id)} := map(\overline{pair_{t,t'}} \times id)$.

For the complete proof however, we need to do the induction on a stronger hypothesis, namely: $\forall p \geq 1$, $\forall \bar{\sigma}_1, \ldots, \bar{\sigma}_p$ such that $erase(\bar{\sigma}_1) = \ldots = erase(\bar{\sigma}_p)$, there are some q tagged types $\bar{\tau}_1, \ldots \bar{\tau}_q$ and q tagged functions $\bar{f}_k : \bar{\sigma}_1 \times \ldots \times \bar{\sigma}_p \rightarrow \bar{\tau}_k$, $k = 1, q$, such that $[\![f]\!]_{\mathcal{B}\cup\mathcal{I}}(x_1 \cup \ldots \cup x_p) = \bigcup_{l=1,q} [\![\bar{f}]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}(x_1, \ldots, x_p)$, whenever $x_k \in [\![\bar{\sigma}_k]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}$, $\forall k = 1, p$. Then composition becomes straightforward. The only complicated case is $bfix(f \mid g) : \sigma \rightarrow \tau$. We first apply induction hypothesis to g, to get $\bar{g}_l : \bar{\sigma}_1 \times \ldots \times \bar{\sigma}_p \rightarrow \bar{\tau}_l$, $l = 1, q$. Then apply again induction hypothesis for f, by forcing $\bar{\sigma}_1 \times \ldots \times \bar{\sigma}_p \times \bar{\tau}_1 \times \ldots \times \bar{\tau}_q$ as its input, to get q' tagged functions $\bar{f}_l : \bar{\sigma}_1 \times \ldots \times \bar{\sigma}_p \times \bar{\tau}_1 \times \ldots \times \bar{\tau}_q \rightarrow \bar{\tau}'_l$, $l = 1, q'$[14]. Now we use the fact that, for the bounded fixpoint, it is the intersection $f \cap g$ that matters, and so we select only the tagged types $\bar{\tau}_{l_1}, \ldots \bar{\tau}_{l_{q''}}$ which also occur in $\bar{\tau}'_1, \ldots, \bar{\tau}'_{q'}$ (so $q'' \leq min(q, q')$). Finally, replace all arguments in $\bar{f}_{l_i}$, $i = 1, q''$, correspoding to types $\bar{\tau}_l$ which *are not* among $\bar{\tau}_{l_1}, \ldots \bar{\tau}_{l''_q}$, with ϕ, and call $\bar{f}'_{l_i}$ the resulting function. This gives us q'' functions $\bar{f}'_{l_i} : \bar{\sigma}_1 \times \ldots \times \bar{\sigma}_p \times \bar{\tau}_{l_1} \times \ldots \times \bar{\tau}_{l_{q''}} \rightarrow \bar{\tau}_{l_i}$. Take $bfix((\bar{f}'_{l_1}, \ldots, \bar{f}'_{l_{q''}}) \mid (\bar{g}_{l_1}, \ldots \bar{g}_{l_{q''}}))$: its q'' projections form the desired tagging of $bfix(f \mid g)$. □

Notice that, when we apply this lemma to our $Q_f : \pi_\sigma \rightarrow \tau$, we get all taggings of τ to be equal to τ itself ! (Because τ doesn't contain any occurrence of the index I). So we may construct the union of the tagged functions *in the language*, to get some $\bar{Q}_f : \bar{\pi}_\sigma \rightarrow \tau$ such that $[\![Q_f]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}(x) = [\![\bar{Q}_f]\!]_{\mathcal{B}\cup\mathcal{I}}(x)$, for all $x \in [\![\pi_\sigma]\!]_{\mathcal{B}\cup\bar{\mathcal{I}}}$.

[14] To be precise, we should enforce $\Pi_{k=1,p,l=1,q}(\bar{\sigma}_k \times \bar{\tau}_l)$ as the tagged input type for f, and obtain, by induction, some functions of type $\Pi_{k=1,p,l=1,q}(\bar{\sigma}_k \times \bar{\tau}_l) \rightarrow \bar{\tau}'_l$. But then, we compose these functions, to the right, with the obvious grouping function $\bar{\sigma}_1 \times \ldots \times \bar{\sigma}_p \times \bar{\tau}_1 \times \ldots \times \bar{\tau}_q \rightarrow \Pi_{k=1,p,l=1,q}(\bar{\sigma}_k \times \bar{\tau}_l)$.

8 Eliminating the tagged indexes

Now we define a second translation, $\widetilde{(\ )}$ from $\mathcal{RA}(\Sigma \cup \bar{I}) + bfix$ to $\mathcal{RA}(\Sigma) + bfix$. For this, fix some m scalar types $t_1, \ldots, t_m$[15], and translate any tagged type $\bar{\sigma}$ to $\tilde{\bar{\sigma}}$, by replacing occurrences of $\bar{I}_{\tau_k}$ with t_k, occurrences of $\bar{I}_{LEFT}$ or $\bar{I}_{RIGHT}$ with $unit$, and $\bar{I}_{<t,t'>}$ with $\tilde{\bar{I}}_t \times \tilde{\bar{I}}_{t'}$. Then, translate each expression $\bar{f} : \bar{\sigma} \to \bar{\tau}$, by replacing every occurrence of $pair_{t,t'}$, $left$ or $right$, with the identity of the appropriate type.

In the same spirit, we define a tagged pair-index structure from an ordinary model $\mathcal{B}$. Consider m finite sets $x_k \subseteq [\![t_k]\!]_{\mathcal{B}}$, $k = 1, m$ (i.e. $x_k \in [\![\{t_k\}]\!]_{\mathcal{B}}$); we define the tagged index pair structure $\bar{\mathcal{I}}^x$ generated by x ($x = (x_1, \ldots, x_m)$) to be: $\bar{\mathcal{I}}^x_{\tau_k} := x_k$, $\bar{\mathcal{I}}^x_{LEFT} = \bar{\mathcal{I}}^x_{RIGHT} := \{()\}$, $\bar{\mathcal{I}}^x_{t,t'} := \bar{\mathcal{I}}^x_t \times \bar{\mathcal{I}}^x_{t'}$, and $Left, Right, Pair_{t,t'}$ to be the identities of appropriate types. Clearly, $[\![\bar{\sigma}]\!]_{\mathcal{B} \cup \bar{\mathcal{I}}^x} \subseteq [\![\tilde{\bar{\sigma}}]\!]_{\mathcal{B}}$, for each tagged type $\bar{\sigma}$. The following lemma is trivial:

Lemma 5 (Soundness of the $\widetilde{(\ )}$ translation) *For every $\bar{f} : \bar{\sigma} \to \bar{\tau}$ in $\mathcal{RA}(\Sigma \cup \bar{I})$, $\forall z \in [\![\bar{\sigma}]\!]_{\mathcal{B} \cup \bar{\mathcal{I}}^x}$, $[\![\bar{f}]\!]_{\mathcal{B} \cup \bar{\mathcal{I}}^x}(z) = [\![\tilde{\bar{f}}]\!]_{\mathcal{B}}(z)$.*

9 Proof of the conservativity theorem

To conclude the proof of theorem 2, we first establish:

Proposition 10 *When all function symbols in Σ have set height 0, then $\mathcal{NRA}(\Sigma) + bfix$ is a conservative extension of $\mathcal{RA}(\Sigma) + bfix$.*

Proof Consider some function $f : \sigma \to \tau$, where $\sigma = \{t_1\} \times \ldots \times \{t_m\}$, and construct $Q_f : \pi_\sigma \to \tau$. An input for f is an m-tuple $x = (x_1, \ldots, x_m)$ of sets, $x_k : \{t_k\}$. Any encoding of x is an m-tuple $r = (r_1, \ldots, r_m)$, with $r_k : \{I\} \times [I \to \pi_{t_k}]$. We pick some particular encoding, namely we choose m disjoint sets of atoms $\bar{\mathcal{I}}_{\tau_k} \subseteq \mathcal{I}$, $k = 1, m$, which don't contain $Left$ or $Right$, such that $card(\bar{\mathcal{I}}_{\tau_k}) = card(x_k)$. Then $r_k = rel(\psi_k)$[16], where ψ_k is some bijection from $\bar{\mathcal{I}}_{\tau_k}$ to the set of encodings of the elements of x_k. To be precise, for some bijection $\omega_{t_k} : \bar{\mathcal{I}}_{\tau_k} \to x_k$, we have $\omega_{t_k} \sim \psi_k$, $\forall k = 1, m$. The collection $(\bar{\mathcal{I}}_{\tau_1}, \ldots, \bar{\mathcal{I}}_{\tau_m})$ *generates* a tagged pair-index structure embedded in $\mathcal{I}$, $\bar{\mathcal{I}} \sqsubseteq \mathcal{I}$. After tagging $Q_f : (\{I\} \times [I \to \pi_{t_1}]) \times \ldots \times (\{I\} \times [I \to \pi_{t_m}]) \to \tau$, like in lemma 4, we get $\bar{Q}_f : (\{\bar{I}_{\tau_1}\} \times [\bar{I}_{\tau_1} \to \pi_{t_1}]) \times \ldots \times (\{\bar{I}_{\tau_m}\} \times [\bar{I}_{\tau_m} \to \pi_{t_m}]) \to \tau$ for which $[\![Q_f]\!]_{\mathcal{B} \cup \mathcal{I}}(r_1, \ldots, r_m) = [\![\bar{Q}_f]\!]_{\mathcal{B} \cup \bar{\mathcal{I}}}(r_1, \ldots, r_m)$.

The next step is to eliminate the tagged indexes from $\bar{Q}_f$, using the technique in lemma 5, and get $\tilde{\bar{Q}}_f : (\{t_1\} \times [t_1 \to t_1]) \times \ldots \times (\{t_m\} \times [t_m \to t_m])$. From our input x, we construct the tagged index-pair structure $\bar{\mathcal{I}}^x$ (like in lemma 5), for which the functions $[\![\bar{Q}_f]\!]_{\mathcal{B} \cup \bar{\mathcal{I}}^x}$ and $[\![\tilde{\bar{Q}}_f]\!]_{\mathcal{B}}$ coincide on $[\![\bar{\sigma}]\!]_{\mathcal{B} \cup \bar{\mathcal{I}}^x}$.

Now we make the key observation that the tagged index-pair structures $\bar{\mathcal{I}}$ and $\bar{\mathcal{I}}^x$ are isomorphic, because they have finite sets of the same cardinality at the base tags ! Recall that $\bar{\mathcal{I}}^x_{\tau_k} = x_k$; then $\omega_{t_k} : \bar{\mathcal{I}}_{\tau_k} \to x_k$ is the desired (canonical) bijection, and it extends to a family of isomorphisms $\omega_\nu : [\![\nu]\!]_{\mathcal{B} \cup \bar{\mathcal{I}}} \to [\![\nu]\!]_{\mathcal{B}}$ at each type ν (which is the identity at τ, because τ doesn't contain any index type). So, to compute $[\![Q_f]\!]_{\mathcal{B} \cup \mathcal{I}}(r_1, \ldots, r_m)$, we simply compute $[\![\tilde{\bar{Q}}_f]\!]_{\mathcal{B}}(\omega_{\{t_1\}}(r_1), \ldots, \omega_{\{t_m\}}(r_m))$. This is trivially done, because the isomorphisms ω_{t_k} are "canonical", so $\omega_{\{t_k\}}(r_k) : \{t_k\} \times [t_k \to t_k]$ can be uniquely computed from x_k (it doesn't depend on the choice of the indexes in $\bar{\mathcal{I}}_{\tau_k}$): namely, $\omega_{\{t_k\}}(r_k) = (x_k, ID_{x_k})$, where ID_{x_k} is the function with finite support (viewed as a relation) assigning to each value $z \in x_k$ (which is a scalar, i.e. tuple of basic values $z = (z_1, z_2, \ldots)$) its encoding (which is $(\{z_1\}, \{z_2\}, \ldots)$). Clearly, the function $x_k \rightsquigarrow (x_k, ID_{x_k})$ is computable in $\mathcal{RA}$, and this concludes our proof. □

[15] Later, we pick them from the input types of our function $f : \{t_1\} \times \ldots \times \{t_m\} \to \{t\}$ to be translated.
[16] Recall that $rel(\psi)$ is the encoding, as a tuple of relations, of some partial, finite function ψ.

To obtain the desired result for $\mathcal{NRA}_1(\Sigma)$, we make the following observation, which is easily proven by induction on the structure of f:

Lemma 6 *If all function symbols in Σ have set height 0 and $f : s \times \sigma \to t$ is in $\mathcal{NRA}(\Sigma)$ (s, t scalar types, σ a flat type), then there is some $g : s \to t$ in $\mathcal{NRA}_0(\Sigma)$, such that $f = g \circ \pi_1$.*

Proposition 11 *If all function symbols in Σ have set height 0 then $\mathcal{NRA}(\Sigma)$ is a conservative extension of $\mathcal{NRA}_1(\Sigma)$*

Proof It suffices to consider two kinds of functions in $\mathcal{NRA}(\Sigma)$: $f : s \times \sigma \to t$ and $g : s \times \sigma \to \{t\}$, with s, t scalar types and σ a flat type. The previous lemma takes care of f. For the second function, we write g as $s \times \sigma \xrightarrow{\eta \times id} \{s\} \times \sigma \xrightarrow{ext_1(g)} \{t\}$. By the previous proposition, $ext_1(g)$ is in $\mathcal{RA}(\Sigma)$, so g is in $\mathcal{NRA}_1(\Sigma)$. $\square$

10 How to extend the conservativity theorem in the presence of external functions of set height 1

Function symbols in Σ of set height 1 may be, essentially, of two kinds (s, s' are scalar types, σ is some flat type):

Aggregate functions $f : \sigma \to s$ (or $f : s' \times \sigma \to s$).

Set constructors $f : s \to \{s'\}$, or $f : s \times \sigma \to \{s'\}$.

It is not too difficult to define a reasonable relational algebra in this case: instead of the **projections and scalar functions** and **selections** rules in the definition of $\mathcal{RA}(\Sigma, \mathcal{X})$, simply add the following two rules:

ext For every $f : s \to \{s'\}$ in $\mathcal{NRA}_1(\Sigma)$, $ext(f) : \{s\} \to \{s'\}$ is in $\mathcal{RA}(\Sigma, \mathcal{X})$.

ext$_1$ For every $f : s \times \sigma \to \{s'\}$ in $\mathcal{NRA}_1(\Sigma)$, $ext_1(f) : \{s\} \times \sigma \to \{s'\}$ is in $\mathcal{RA}(\Sigma, \mathcal{X})$.

Because $\mathcal{NRA}_1(\Sigma)$ was a conservative extension of $\mathcal{RA}(\Sigma)$, when all function symbols in Σ had set height 0, we can argue that this language is a reasonable extension of the relational algebra, for external functions of height 1.

The two map lemmas, propositions 8 and 9, still hold (note that $ext_1(f)$ is not strict in its first argument): $[t \to ext(f)] := ext(id_t \times f)$, and $[t \Rightarrow ext_1(f)] := (\pi_1, ext_1(g) \circ \pi_2)$, where $g : (t \times s) \times [t \to \sigma] \to \{t \times s'\}$ is $g(i, x, \varphi) := \{i\} \bowtie f(x, ap_\sigma(\varphi, i))$. It is also easy to translate an aggregate function $f : \sigma \to s$ (or $f : s' \times \sigma \to s$) into $R_f : \pi_\sigma \to \{I \times s\}$ (or $R_f : \pi_{s'} \times \pi_\sigma \to \pi_s$). The difficulties arise when we try to translate a set constructor $f : t \to \{s\}$ (or $f : t \times \sigma \to \{s\}$): then R_f needs to invent indexes for the set it constructs ! We handle this by adding function symbols $c_s : \{s\} \to \{I \times s\}$ for "inventing" indexes, like in [23], to the extended signature $\Sigma \cup I$ (one c_s for each scalar type s which is the codomain of some set constructor in Σ), such that $c_s(x)$ simply chooses some index for each element in x. For a given atomic pair-index structure $\mathcal{I}$, we consider a family of infinite sets of atoms $\mathcal{A}_s$ (s a scalar type), s.t. $\mathcal{A}_s \cap \mathcal{A}_{s'} = \phi$ (when $s \neq s'$), $Left, Right \notin \mathcal{A}_s$ and there remain infinitely many atoms outside of $\bigcup \mathcal{A}_s$. Then, we require the interpretation of c_s to return only atoms in $\mathcal{A}_s$. We add $< s >$ to the basic tags, for each scalar type s, and we tag $c_s : \{s\} \to \{I \times s\}$ as $\bar{c}_s : \{s\} \to \{\bar{I}_{<s>} \times s\}$. When we eliminate the tags, we encode $\bar{I}_{<s>}$ by s, $\bar{\mathcal{I}}^x_{<s>} := [\![s]\!]_{\mathcal{B}}$, and we translate $\bar{c}_s$ by $\tilde{\bar{c}}_s(y) := \{(z, z) \ / z \in y\}$. Finally, when we encode the inputs $x_1, \ldots, x_m$ (see the proof of proposition 10), we choose atomic indexes outside of $\bigcup \mathcal{A}_s$.

11 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, in the presence of some order relation on its inputs. The main result consists in proving that, at flat types, the language coincides with well known languages, like first order logic with (inflationary or monotone) fixpoints (or $DATALOG^{\neg}$) - in the inflationary case - or the *while* queries (or the first order logic with partial fixpoints, or $DATALOG^{*\neg}$), in the case of partial fixpoints.

The technique developed for proving the conservativity result, can be used to prove a (weaker) conservativity result for unbounded fixpoints. Indexes have been used before, for a similar conservativity result (without fixpoints), but in the presence of inventions. Here, we make the observation that two constants and a binary function over the set of indexes are all one needs to encode operations of higher types. These operations can be simply viewed as any other functions in the signature Σ, so that indexes need not be treated as special extensions of the language.

The *pair* function over an index set I should be viewed in a different way than the *interpreted* function symbols in [1]. It's interpretation is not fixed apriori, but the computing device for some query may interrogate *pair* as an oracle. As a consequence, we would like to view the query $f : I \to \{I\}$, $f(x) = \{y \,/ pair(y, y) = x\}$ as *not* being domain independent, because it depends on the particular interpretation of *pair*, and cannot be answered in finite time, by interrogating *pair* as an oracle. We intend to investigate these aspects in the future.

12 Acknowledgements

I wish give special thanks to Val Breazu-Tannen, for numerous discussions and comments, which made this work possible. I am deeply grateful to Peter Buneman, for his suggestions and encouragement. 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*, MS-CIS-92-47, University of Pennsylvania, 1992

[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] Y. Gurevich, S. Shelah, *Fixed-point extensions of first-order logic.* Annals of Pure and Applied Logic, 32:265-280, 1986

[14] R. Hull, J. Su, *On Accessing Object-Oriented Databases: Expressive Power, Complexity, and Restrictions*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989

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

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

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

[18] L. Libkin, L. Wong, *Query Languages for Bags*, TR, University of Pennsylvania, 1993

[19] S. MacLane, *Categories for the working mathematician*, Springer-Verlag, Berlin,1972

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

[21] P. Trinder, *Comprehensions, a Query Notation for DBPLs*, Proc. of 3rd International Workshop on Database Programming Languages, 1990

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

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

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

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

[26] P. Wadler, *Comprehending Monads*, Proc of ACM Conference on Lisp and Functional Programming, 1990

[27] P Wadler, *Notes on monads and ringads*, personal notes, personal notes, 1990

[28] D. Watt, P. Trinder, *Towards a Theory of Bulk Types*, Fide Technical Report 91/26, Glasgow University, Glasgow G12 8QQ, 1991

[29] L. Wong, *A Conservative Property of a Nested Relational Query Language*, MS-CIS-92-59, University of Pennsylvania, 1992