CORE
Metadata, citation and similar papers at core.ac.uk
Provided by ScholarlyCommons@Penn

University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS) Department of Computer & Information Science

July 1992

A Conserative Property of a Nested Relational Query Language

Limsoon Wong
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Recommended Citation
Limsoon Wong, "A Conserative Property of a Nested Relational Query Language", . July 1992.

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-59.

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/472
For more information, please contact repository@pobox.upenn.edu.

A Conserative Property of a Nested Relational Query Language

Abstract

We proposed in [7] a nested relational calculus and a nested relational algebra based on structural recursion [6,5] and on monads [27,16]. In this report, we describe *relative set abstraction* as our third nested relational query language. This query language is similar to the well known list comprehension mechanism in functional programming languages such as Haskell [Il], Miranda [24], KRC [23], etc. This language is equivalent to our earlier query languages both in terms of semantics and in terms of equational theories. This strong sense of equivalence allows our three query languages to be freely combined into a nested relational query language that is robust and user-friendly.

Comments

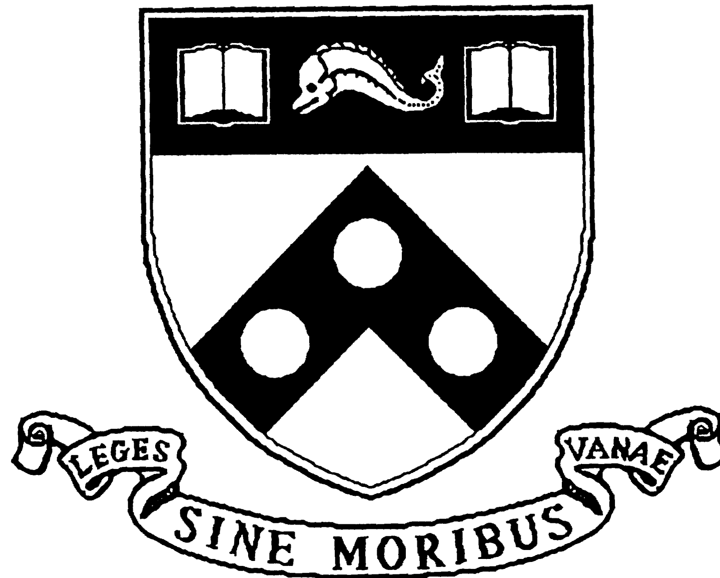
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-59.

This technical report is available at ScholarlyCommons: https://repository.upenn.edu/cis_reports/472

A Conservative Property Of A Nested Relational Query Language

MS-CIS-92-59
LOGIC & COMPUTATION 48

Limsoon Wong



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

July 1992

A Conservative Property of a Nested Relational Query Language *

Limsoon Wong

Department of Computer and Information Science
University Of Pennsylvania
Philadelphia, PA 19104-6389, USA

1 Summary

We proposed in [7] a nested relational calculus and a nested relational algebra based on structural recursion [6, 5] and on monads [27, 16]. In this report, we describe *relative set abstraction* as our third nested relational query language. This query language is similar to the well known list comprehension mechanism in functional programming languages such as Haskell [11], Miranda [24], KRC [23], etc. This language is equivalent to our earlier query languages both in terms of semantics and in terms of equational theories. This strong sense of equivalence allows our three query languages to be freely combined into a nested relational query language that is robust and user-friendly.

Every expression of relative set abstraction can be reduced to a normal form. This normal form has an immediately apparent property that is very interesting: *an expression in normal form does not have any subexpression with set height exceeding the set height of the type of the expression.* One way to view this result is to classify expressions of the language into a hierarchy of sublanguages L_0, L_1, ..., where L_n consists of expressions of height not exceeding n. Then every expression in L_m whose type has height n is equivalent to an expression in L_n. Consequently, L_m is a *general conservative extension* of all L_n where $n < m$.

In particular, one can obtain the conservativeness of relative set abstraction with respect to the traditional flat relational algebra from this observation. That is, if an expression denotes a function from several flat relations to a flat relation, then this function is expressible in the traditional flat relational algebra. The converse that every function definable by flat relational algebra is definable using relative set abstraction is also true. The connection is therefore very tight.

These results have several consequences that are very easy to derive. From the existence of normal form of definable functions, one can readily check that conditional on base type is not definable as there is no normal form that defines it. Similarly, the existence of normal form can be used to show that every definable function of type $\{s\} \to b$ must be a constant function. From the fact that our language is

*Extended abstract to appear as "Normal forms and conservative properties for query languages over collection types" in the *Proceedings of PODS'93.*

a conservative extension of flat relational algebra, we know that it cannot express transitive closure. This latter observation has an immediate consequence: transitive closure cannot be expressed in the language proposed by Abiteboul and Beeri without using the powerset operator.

Due to the robust connection between relative set abstraction and our two earlier languages, all the results mentioned above also hold for these languages. That is, they hold for nested relational calculus and for nested relational algebra. Hence we have filled in some of the gaps left by other researchers. To begin with, Paredaens and Van Gucht [17, 18] showed that the nested relational algebra of Thomas and Fischer [20] is conservative with respect to flat relational algebra in the sense we have described. The Thomas and Fischer algebra is a very restricted query language where all operators can be applied only to the topmost level of relations. Our result extends Paredaens and Van Gucht's to a richer language. Hull and Su proposed a nested relational query language in which powerset is expressible and studied its expressive power [12]. One of their result is that it is not conservative with respect to the flat relational algebra in our sense. Grumbach and Vianu proved in [8] that the language of Hull and Su is not conservative with respect to set height of input/output at all. In constrast, our language cannot express powerset and is conservative with respect to set height of input/output. This helps clarify the role of powerset in the expressive power of query languages.

The general conservative extension result can be further improved in two ways. Firstly, Many modern data models possess an additional data structuring mechanism known variously as coproducts, variant types, sum types, or tagged unions (see Abiteboul and Hull [3] and Hull and Yap [13] for example). However, many papers on expressive power excluded this feature from consideration [12, 8, 2]. We extend the nested relational calculus of [7] with variant types and prove that the extended calculus remains conservative with respect to height of input/output.

Secondly, the proof we give for relative set abstraction relies on a set-based semantics. This is in line with the work of many researchers as reported in Abiteboul et. al. [1], Abiteboul and Beeri [2], Hull and Su [12], Grumbach and Vianu [8], Paredaens and Van Gucht [17, 18], Gyssens and Van Gucht [10], etc. But our languages can be given interpretations based on bags and lists too. It is desirable to know whether our main result holds when the languages are given list- and bag- semantics. We prove that it does. Moreover, the proof is uniform across these semantics.

The organisation of the remainder of this Report is as follows. Section 2 introduces relative set abstraction and the nested relational calculus of our earlier paper [7]. We establish translations between these languages that preserve semantics, preserve set heights, and preserve and reflect equational theories. Section 3 presents our main result that our query language is conservative with respect to set height of input/output. The general conservativeness result is specialised to conservativeness with respect to flat relational algebra in Section 4. In section 5 we present some easy-to-prove corollaries. The two improvements to the main theorem mentioned above are presented in the final section.

2 Relative set abstraction as a nested relational language

Wadler and Trinder argued that list/set/bag comprehension is a natural query notation [22, 21, 28]. They also demonstrated that this notation does not hamper query optimization. In this section we present a query language based on comprehension that is equivalent to our nested relational algebra

and nested relational calculus. We call this query language *Relative Set Abstraction* (or $\mathcal{RSA}$ for short).

Types. A type in relative set abstraction is either an object type s or is a function type $s \to t$ where s and t are both object type. The object types are given by the grammar:

$$s, t ::= unit \mid b \mid s \times t \mid \{s\}$$

Expressions. The expressions of relative set abstraction are formed according to the rules below. Note that the lexical ordering of $x_1 \in e_1, \ldots, x_n \in e_n$ in $\{e \mid x_1 \in e_1, \ldots, x_n \in e_n\}$ is significant. It must be pointed out that $x_i \in e_i$ is *not* a set membership test. It is the introduction of a variable binding, similar to that of lambda abstraction $\lambda x.e$. It is to emphasize this point that we call this language *relative set abstraction*. We use the notation Δ as a shorthand for $x_1 \in e_1, \ldots, x_n \in e_n$. The scope of a set abstraction variable x_i in $\{e \mid \Delta, x_i \in e_i, \Delta'\}$ is Δ' and e. We adopt the usual convention that distinct variable bindings be given distinct variables. Type superscripts are usually omitted, since they can be inferred. As in [7], booleans are represented by the two values of type $\{unit\}$, with $\{()\}$ for true and $\{\}$ for false. Equality test eq_b is restricted to base type b.

$$\frac{}{x^s : s} \qquad \frac{}{() : unit} \qquad \frac{e : t}{\lambda x^s.e : s \to t} \qquad \frac{e_1 : s \to t \qquad e_2 : s}{e_1\ e_2 : t}$$

$$\frac{e_1 : s \qquad e_2 : t}{(e_1, e_2) : s \times t} \qquad \frac{e : s \times t}{\pi_1\ e : s} \qquad \frac{e : s \times t}{\pi_2\ e : t} \qquad \frac{}{\{\}^s : \{s\}}$$

$$\frac{e_1 : \{s\} \qquad e_2 : \{s\}}{e_1 \cup e2 : \{s\}} \qquad \frac{}{not : \{unit\} \to \{unit\}} \qquad \frac{}{eq_b : b \times b \to \{unit\}}$$

$$\frac{e : s}{\{e\} : \{s\}} \qquad \frac{e_1 : \{s_1\} \quad \ldots \quad e_n : \{s_n\} \qquad e : t}{\{e \mid x_1^{s_1} \in e_1, \ldots, x_n^{s_n} \in e_n\} : \{t\}} \qquad \frac{}{c : b}$$

Semantics. The intended semantics is that of sets and functions. A detailed specification is omitted; but for the Reader who is familiar with the usual notations of denotational semantics [19, 9],

$$[\![\{e \mid x_1 \in e_1, \ldots, x_n \in e_n\}]\!]\rho = \{[\![e]\!]\rho[d_1/x_1, \ldots, d_n/x_n] \mid d_1 \in [\![e_1]\!]\rho \,\&\, \ldots \,\&\, d_n \in [\![e_n]\!]\rho[d_1/x_1, \ldots, d_{n-1}/x_{n-1}]\}$$

Examples. $\{(x, y) \mid x \in X, y \in Y\}$ denotes the cartesian product of the sets denoted by X and Y. $\{y \mid x \in X, y \in x\}$ denotes the flattenning of the set denoted by X.

Equational Theory. We do not have a complete set of axioms yet. However, for the purpose of this report, we treat *not* and eq_b as uninterpreted constants and use the axioms of our nested relational calculus [7] with the following changes: drop all the rules concerning the $\bigcup\{e' \mid x \in e\}$ construct (to be described shortly) and add the rules below.

$$\frac{}{\{e \mid \Delta_1, x \in \{\}, \Delta_2\} = \{\}} \qquad \frac{}{\{e \mid \Delta_1, x \in \{e'\}, \Delta_2\} = \{e[e'/x] \mid \Delta_1, \Delta_2[e'/x]\}}$$

$$\frac{}{\{x \mid x \in e\} = e} \qquad \frac{}{\{e \mid \Delta_1, x \in \{e' \mid \Delta\}, \Delta_2\} = \{e[e'/x] \mid \Delta_1, \Delta, \Delta_2[e'/x]\}}$$

$$\overline{\{e \mid \Delta_1, x \in e_1 \cup e_2, \Delta_2\} = \{e \mid \Delta_1, x \in e_1, \Delta_2\} \cup \{e \mid \Delta_1, x \in e_2, \Delta_2\}}$$

This simple language is equivalent to our nested relational calculus. The remainder of this section is devoted to working out the translations. First let us sketch the calculus of [7] (or $\mathcal{NRC}$ for short). We have also taken the liberty of using the more suggestive syntax $\bigcup\{e' \mid x \in e\}$ in place of the $ext(\lambda x.e')(e)$ syntax of [7]. The reader is referred to [7] for a more detail account.

Types. Same as relative set abstraction.

Expressions. Same as relative set abstraction but replace the $\{e \mid \Delta\}$ construct by the construct $\bigcup\{e' \mid x \in e\}$ whose typing rule is

$$\frac{e : \{s\} \qquad e' : \{t\}}{\bigcup\{e' \mid x^s \in e\} : \{t\}}$$

Semantics. The intended semantics of $\bigcup\{e' \mid x \in e\}$ is to flat-map the function denoted by $\lambda x.e'$ over the set denoted by e. That is, for the Reader who is familiar with standard notations of denotational semantics [19, 9],

$$[\![\bigcup\{e' \mid x \in e\}]\!]\rho = \bigcup_{d \in [\![e]\!]\rho} [\![e']\!]\rho[d/x]$$

Examples. $\bigcup\{\bigcup\{\{(x,y)\} \mid x \in X\} \mid y \in Y\}$ denotes the cartesian product of the sets denoted by X and Y. $\bigcup\{\bigcup\{\{(\pi_1\ x, y)\} \mid y \in \pi_2\ x\} \mid x \in X\}$ denotes the unnesting of the set denoted by X. Finally, to project the left component of pairs inside a set of sets X, $\bigcup\{\{\bigcup\{\pi_1\ y \mid y \in x\}\} \mid x \in X\}$. The last example is noteworthy because it demonstrates "nested projection."

Equational Theory. We list the axioms concerning $\bigcup\{e' \mid x \in e\}$ below.

$$\overline{\bigcup\{e' \mid x \in \{e\}\} = e'[e/x]} \qquad \overline{\bigcup\{\{x\} \mid x \in e\} = e}$$

$$\overline{\bigcup\{e_1 \mid x_1 \in \bigcup\{e_2 \mid x_2 \in e_3\}\} = \bigcup\{\bigcup\{e_1 \mid x_1 \in e_2\} \mid x_2 \in e_3\}}$$

$$\overline{\bigcup\{e \mid x \in e_1 \cup e_2\} = \bigcup\{e \mid x \in e_1\} \cup \bigcup\{e \mid x \in e_2\}} \qquad \overline{\bigcup\{e' \mid x \in \{\}\} = \{\}}$$

$$\overline{\bigcup\{e_1 \cup e_2 \mid x \in e\} = \bigcup\{e_1 \mid x \in e\} \cup \bigcup\{e_2 \mid x \in e\}} \qquad \overline{\bigcup\{\{\} \mid x \in e\} = \{\}}$$

Having introduced the languages, it is time to show that they are equivalent. In fact, we prove that they are equal both in terms of semantics and in terms of equational theories. To this end, we need a translation $\mathcal{NR}[\cdot]$ taking an expression $e : s$ of $\mathcal{NRC}$ to an expression $\mathcal{NR}[e] : s$ of $\mathcal{RSA}$ and a translation $\mathcal{RN}[\cdot]$ taking an expression $e : s$ of $\mathcal{RSA}$ to an expression $\mathcal{RN}[e] : s$ of $\mathcal{NRC}$. The translations are straight forward. The only non-trivial rules are:

$$\frac{\mathcal{NR}[e_1] = e_1' \qquad \mathcal{NR}[e_2] = e_2'}{\mathcal{NR}[\bigcup\{e_1 \mid x \in e_2\}] = \{y \mid x \in e_2', y \in e_1'\}} \text{ where } y \text{ is fresh.}$$

$$\frac{\mathcal{RN}[e] = e' \qquad \mathcal{RN}[e_1] = e_1' \qquad \mathcal{RN}[e_n] = e_n'}{\mathcal{RN}[\{e \mid x_1 \in e_1, ..., x_n \in e_n\}] = \bigcup\{... \bigcup\{\{e'\} \mid x_n \in e_n'\} \mid ... \} \mid x_1 \in e_1'\}}$$

Then relative set abstraction and nested relational calculus are equivalent.

2.1 Theorem

- *Every closed e of $\mathcal{NRC}$ denotes the same value as $\mathcal{NR}[e]$.*
- *Every closed e of $\mathcal{RSA}$ denotes the same value as $\mathcal{RN}[e]$.* □

Moreover, the translations preserve and reflect the equational theories of these languages.

2.2 Theorem

- $\mathcal{RSA} \vdash \mathcal{NR}[\mathcal{RN}[e]] = e$
- $\mathcal{NRC} \vdash \mathcal{RN}[\mathcal{NR}[e]] = e$
- $\mathcal{NRC} \vdash e_1 = e_2$ *if and only if* $\mathcal{RSA} \vdash \mathcal{NR}[e_1] = \mathcal{NR}[e_2]$
- $\mathcal{RSA} \vdash e_1 = e_2$ *if and only if* $\mathcal{NRC} \vdash \mathcal{RN}[e_1] = \mathcal{RN}[e_2]$. □

Since our nested relational calculus does not have membership test, or anything that looks like a nesting operation, we show that they are definable. In [7], it was established that equality test eq_s on all object type s can be used to simulate membership test, subset test, set difference, set intersection, and relational nesting. Therefore, it suffices for us to prove that eq_s is definable in $\mathcal{RSA}$ for all s.

2.3 Proposition

Equality at all types are definable in $\mathcal{RSA}$.

Proof. Let eq_s be the equality test at type s. It can be defined by induction on s.

- $eq_{unit} = \lambda x.\{()\}$
- eq_b is given.
- $eq_{s \times t} = \lambda x.\{() \mid u \in eq_s(\pi_1(\pi_1(x)), \pi_1(\pi_2(x))), v \in eq_t(\pi_2(\pi_1(x)), \pi_2(\pi_2(x)))\}$
- $eq_{\{s\}} = \lambda x.\{() \mid u \in not(\{() \mid q \in \pi_1(x), z \in not(\{() \mid p \in \pi_2(x), w \in eq_s(p,q)\})\}), v \in not(\{() \mid q \in \pi_2(x), z \in not(\{() \mid p \in \pi_1(x), w \in eq_s(p,q)\})\})\}$ □

It then follows from the equivalence between $\mathcal{NRC}$ and the nested relational algebra of [7] (denoted by $\mathcal{NRA}$) that relative set abstraction is also equivalent to $\mathcal{NRA}$.

3 Every definable function is definable using operators whose set height is atmost the set height of the input/output of the function

In this section the main result of the report is proved. The content of which is given in the section title above. Let us first explain what the theorem is about. The *set height* $ht(s)$ of a type s is defined by induction on the structure of type:

- $ht(unit) = ht(b) = 0$
- $ht(s \times t) = ht(s \to t) = \max\{ht(s), ht(t)\}$
- $ht(\{s\}) = 1 + ht(s)$

Note that every expression of our languages has a unique typing derivation. Then the set height of expression e is defined simply as $ht(e) = \max\{ht(s) \mid s \text{ occurs in the type derivation of } e\}$. Then the theorem expresses a very general conservative property. It says that to process information (that is, input/output) of set height n, no operators whose set height exceeding n is required. In other words, if a function whose input/output has height n is defined by an expression e whose height exceeds n, we can find an alternative expression e' whose height does not exceed n to implement it. More prosaicly, using intermediate expressions of greater height does not increase horsepower.

The proof is straight forward for relative set abstraction after a normal form result is established. Then by showing that the translations between our languages preserve set height, the result is transferred to the calculus and the algebra.

3.1 Theorem (Normal Form)

Every expression of $\mathcal{RSA}$ can be reduced to an expression such that in every subexpression of the form $x \in \{e \mid x_1 \in e_1, \ldots, x_n \in e_n\}$, all the e_i are not of the form $\{e \mid \Delta\}$, $e \cup e$, $\{\}$, or $\{e\}$; and there is no subexpression of the form $(\lambda x.e)e'$, $\pi_1(e,e)$, or $\pi_2(e,e)$. Moreover, if e is reduced to e', then $\mathcal{RSA} \vdash e = e'$.

Proof. Consider the following transformation rules which progressively eliminate subexpressions that do not satisfy the requirement of the theorem. We use $\mathcal{C}[\cdot]$ to stand for a context with a hole and $\mathcal{C}[e]$ for plugging e into the hole of context $\mathcal{C}[\cdot]$.

1. $\mathcal{C}[\{e \mid \Delta_1, x \in e_1 \cup e_2, \Delta_2\}] \leadsto \mathcal{C}[\{e \mid \Delta_1, x \in e_1, \Delta_2\} \cup \{e \mid \Delta_1, x \in e_2, \Delta_2\}]$
2. $\mathcal{C}[\{e \mid \Delta_1, x \in \{e' \mid \Delta'\}, \Delta_2\}] \leadsto \mathcal{C}[\{e[e'/x] \mid \Delta_1, \Delta', \Delta_2[e'/x]\}]$
3. $\mathcal{C}[\{e \mid \Delta_1, x \in \{e'\}, \Delta_2\}] \leadsto \mathcal{C}[\{e[e'/x] \mid \Delta_1, \Delta_2[e'/x]\}]$

4. $\mathcal{C}[\{e \mid \Delta_1, x \in \{\}, \Delta_2\}] \leadsto \mathcal{C}[\{\}]$

5. $\mathcal{C}[(\lambda x.e)e'] \leadsto \mathcal{C}[e[e'/x]]$

6. $\mathcal{C}[\pi_1(e_1, e_2)] \leadsto \mathcal{C}[e_1]$

7. $\mathcal{C}[\pi_2(e_1, e_2)] \leadsto \mathcal{C}[e_2]$

Each transformation corresponds to an axiom in the equational theory of relative set abstraction. The normal forms of this rewriting system clearly satisfy the requirement of the theorem. It remains to show that the rewriting system is terminating. In fact, we prove a stronger property: the rewriting system is strongly normalising.

Let φ maps variable names to a natural number greater than 1. Let $\varphi[n/x]$ be the function which assigns n to x but agrees with φ on other variables. Let $||e||\varphi$, defined below, measures the size of e in the environment φ where each free variable x in e is given the size $\varphi(x)$.

- $||x||\varphi = \varphi(x)$
- $||()||\varphi = ||\{\}||\varphi = ||c||\varphi = ||not||\varphi = ||eq_b||\varphi = 2$
- $||\pi_1\ e||\varphi = ||\pi_2\ e||\varphi = ||\{e\}||\varphi = ||not\ e||\varphi = ||eq_b\ e||\varphi = 1 + ||e||\varphi$
- $||e_1\ \cup\ e_2||\varphi = 1 + ||e_1||\varphi + ||e_2||\varphi$
- $||(e_1, e_2)||\varphi = ||e_1||\varphi + ||e_2||\varphi$
- $||(\lambda x.e)(e')||\varphi = ||e||\varphi[||e'||\varphi/x] + ||e'||\varphi$
- $||\lambda x.e||\varphi = ||e||\varphi[2/x]$
- $||\{e_n \mid x_1 \in e_0, ..., x_n \in e_{n-1}\}||\varphi = ||e_0||\varphi_0 \cdot ... \cdot ||e_n||\varphi_n$, where $\varphi_0 = \varphi$ and $\varphi_{i+1} = \varphi_i[||e_i||\varphi_i/x_{i+1}]$.

A lemma. Let φ_1 and φ_2 be such that for every x, $\varphi_1(x) \leq \varphi_2(x)$. By a routine induction on e, we have $||e||\varphi_1 \leq ||e||\varphi_2$. **A corollary of the lemma.** Let $||e'||\varphi \leq n$. By an induction on e and the previous lemma, we have $||e[e'/x]||\varphi \leq ||e||\varphi[n/x]$. Now it follows readily that for any choice of φ, $e \leadsto e'$ implies $||e||\varphi > ||e'||\varphi$. $\square$

It is now easy to prove the main theorem.

3.2 Theorem (General Conservative Extension)

Let $e : s$ be an expression of $\mathcal{RSA}$. Then there is an expression e' such that $ht(e') \leq \max(\{ht(s)\} \cup \{ht(s) \mid s$ is the object type of a free variable in $e\})$. Moreover, $\mathcal{RSA} \vdash e = e'$.

Proof. Let e' be normal-form of e. Now we verify its height by structural induction on it. Let k be the maximum height of the free variables in e.

Case $e' : s$ is x, $\{\}$, c, eq_b, not, or $()$. Immediate.

Case $e' : s$ is $\{e''\}$. Immediate by hypothesis on e''.

Case $e' : t_1 \times t_2$ is (e_1, e_2). By hypothesis, $ht(e_1) \leq \max\{k, ht(t_1)\}$ and $ht(e_2) \leq \max\{k, ht(t_2)\}$. Then $ht(e') = \max\{k, ht(e_1), ht(e_2)\} \leq \max\{k, ht(s)\}$.

Case $e' : t_1 \to t_2$ is $\lambda x.e''$. By hypothesis, $ht(e'') \leq \max\{k, ht(t_2)\}$. So $ht(e') = \max\{ht(s), ht(e'')\} \leq \max\{k, ht(s)\}$.

Case $e' : s$ is $\pi_1\ e''$ or $\pi_2\ e''$. Then e'' must be a free variable or is a chain of projections on a free variable. The case thus holds.

Case $e' : \{t\}$ is $\{e'' \mid x_1 \in e_1, \ldots, x_n \in e_n\}$. By hypothesis, $ht(e_i) \leq \max\{k, 1+ht(x_1), \ldots, 1+ht(x_{i-1})\}$. Now we show by induction on i that the $1 + ht(x_j)$ can be replaced by 1. Starting with e_1. If e_1 is of the form $not(\cdot)$ or of the form $eq_b(\cdot)$, then $ht(x_1) = 0$. Otherwise, e_1 must be a chain of projections on a free variable, then $ht(x_1) \leq k$. In either case, $ht(e_i) \leq \max\{k, 1, 1+ht(x_2), \ldots, 1+ht(x_{i-1})\}$. The analysis can be repeated for the remaining e_i. Then $ht(e_i) \leq \max\{k, 1\}$. By hypothesis, $ht(e'') \leq \max\{k, ht(t)\}$. Then $ht(e') = \max\{k, ht(s), ht(e''), ht(e_1), \ldots, ht(e_n)\} \leq \max\{k, ht(s)\}$. □

It is straight forward to see that for any e of $\mathcal{NRC}$, $ht(e) = ht(\mathcal{NR}[e])$; and for any e of $\mathcal{RSA}$, $ht(e) = ht(\mathcal{RN}[e])$. Consequently, the above result can be transferred to $\mathcal{NRC}$. The translations given between $\mathcal{NRC}$ and $\mathcal{NRA}$ in [7] do not preserve set height of expressions. However, if we add $ext(f)$, $ext_2(f)$, and $map_2(f)$ to $\mathcal{NRA}$ as primitives and use them to replace $\mu \circ map(f)$, $\mu \circ map(f) \circ \rho_2$, and $map(f) \circ \rho_2$, the translations do preserve set height. Thus the above result also holds for $\mathcal{NRA}$.

As remarked earlier, the above theorem implies height of input/output dictates the kind of functions that our languages can express. In particular, using intermediate expressions of greater heights does not add expressive power. This is in contrast to languages considered by [2, 1, 12, 8] where the kind of functions that can be expressed is not characterised by the height of input/output and is sensitive to the height of intermediate operators. The principal difference between our languages and these languages is that powerset is not expressible in our languages [7] but is expressible in those other languages. This indicates a non-trivial contribution to expressive power by an operation such as a powerset. Although the precise nature of the contribution remains to be characterised.

This result has a practical significance. Some databases are designed to support nested sets up to a fixed depth of nesting. For example, Jaeschke and Schek [14] consider nonfirst normal form relations in which attribute domains are limited to powersets of simple domains (that is, databases whose height is atmost 2). "$\mathcal{RSA}$ restricted to expressions of height 2" is a natural query language for such a database. But knowing that $\mathcal{RSA}$ conservatively such a language, one can instead provide the user with the entire language $\mathcal{RSA}$ as a more convenient query language for this database, so long as queries have input/output height not exceeding 2.

4 Every definable function on flat relations is expressible in flat relational algebra

It is our thesis that if a function definable in our languages has a type that is admissible in a relational database (that is, the height of the type is 1), then this function is definable in the traditional flat relational algebra. This is a consequence of the main theorem. However, as flat relational algebra [15, 4] looks quite different from our languages, it is necessary for us to provide a little more detail. In this section we demonstrate that our languages are indeed conservative with respect to flat relational algebra. Let us introduce a brand of flat relational algebra which we denote by $\mathcal{TRA}$.

Types. The 0-types, corresponding to tuples of atomic values, are given by $O ::= unit \mid b \mid O \times O$. The 1-types, corresponding to tuples of relations, are given by $S ::= \{O\} \mid S \times S$. The query types, corresponding to queries on relations, are all of the form $S \to \{O\}$.

Basic Operators. The basic operators of flat relational algebra are listed (and implemented) below.

- $\Pi : \{O_1 \times O_2\} \to \{O_1\}$. This is the relational project on the left column of a binary relation. It is implemented in $\mathcal{RSA}$ by $\lambda x.\{\pi_1\ y \mid y \in x\}$.
- $cmmt : \{O_1 \times O_2\} \to \{O_2 \times O_1\}$. This swaps the two columns of a binary relation. It is implemented in $\mathcal{RSA}$ by $\lambda x.\{(\pi_2\ y, \pi_1\ y) \mid y \in x\}$.
- $assc : \{O_1 \times (O_2 \times O_3)\} \to \{(O_1 \times O_2) \times O_3\}$. This shifts the tupling brackets around in th obvious way. It is implemented in $\mathcal{RSA}$ by $\lambda x.\{((\pi_1\ y, \pi_1(\pi_2\ y)), \pi_2(\pi_2 y)) \mid y \in x\}$.
- $copy : \{O\} \to \{O \times O\}$. This duplicates the input relation. It is implemented in $\mathcal{RSA}$ by $\lambda x.\{(y, y) \mid y \in x\}$. This operator and the previous three together captures all possible ways of doing relational projections.
- $cartprod : \{O_1\} \times \{O_2\} \to \{O_1 \times O_2\}$. This is the cartesian product. It is implemented in $\mathcal{RSA}$ by $\lambda x.\{(y, z) \mid y \in \pi_1\ x, \pi_2\ x\}$.
- $minus : \{O\} \times \{O\} \to \{O\}$. This 'substracts' the second input from the first. That is, it is the asymmetric set difference operator. It is implemented by in $\mathcal{RSA}$ by $\lambda x.\{y \mid y \in \pi_1\ x, z \in not\ \{() \mid w \in \pi_2\ x, eq_O(y, w)\}\}$.
- $union : \{O\} \times \{O\} \to \{O\}$. This is set union. It already exists in $\mathcal{RSA}$ as $\cup$.
- $select(P_1, P_2) : \{O\} \to \{O\}$, where $P_1 : O \to O'$ and $P_2 : O \to O'$ are expressions constructed from: $P ::= id \mid \langle P, P \rangle \mid P \circ P \mid \pi_1 \mid \pi_2 \mid Kc \mid K()$. It is the selection operator. Intuitively, $select(P_1, P_2)$ is equal to the $\mathcal{NRC}$ expression $ext(\lambda x^O.if\ eq_{O'}(P_1\ x^O, P_2\ x^O)\ then\ \{x\}\ else\ \{\})$. Although the conditional is not a primitive of $\mathcal{NRC}$, it is expressible in $\mathcal{NRC}$ (as it is a conditional on set type) by exploiting the fact that the cartesian product of the empty set with any set is the empty set.
- Other items such as $K\{()\}$ that expresses the constant relation $\{()\}$. They are not very important and we omit the details.

From the foregoing description, it should be clear that every query in $\mathcal{TRA}$ is expressible in $\mathcal{RSA}$, $\mathcal{NRC}$, and $\mathcal{NRA}$ using expression of height 1. It remains for us to sketch a proof of the converse.

4.1 Theorem (Conservative Extension)

Every closed $\mathcal{RSA}$, $\mathcal{NRC}$, and $\mathcal{NRA}$ expression $e : S \to \{O\}$ is expressible in $\mathcal{TRA}$.

Proof (Sketch). By the Normal Form Theorem, it suffices for us to exhibit a translation from such $\mathcal{RSA}$ expression to $\mathcal{TRA}$. In fact, it suffices for us to explain how to express a normalised $E = \{e \mid \Delta\}$ containing a single free variable z of type S in $\mathcal{TRA}$. We do it step by step.

Step 1. Replace all x^{unit} in E by $()$. Let the resulting 'pseudo' expression by E_1.

Step 2. E_1 may be badly formed since it can have subexpression of the form $() \in E'$. Replace each $()$ that appears in this situation by a fresh variable y^{unit}. Let E_2 be the result.

E_2 is now a well formed expression of $\mathcal{RSA}$. Moreover, if y^{unit} appears in E_2, it appears exactly once and only as $y^{unit} \in E'$. Because we started with a normalised expression, such E' must be of the form $proj\ z$, $eq_b\ e'$, or $not\ e'$, where $proj\ z$ is a chain of projections on z.

Step 3. Shuffle the set abstractions in E_2 to get an expression of the form: $\{e \mid x_1 \in proj\ z, ..., x_n \in proj\ z, y'_1 \in eq_b\ e'_1, ..., y'_m \in eq_b\ e'_m, y''_1 \in not\ e''_1, ..., y''_k \in not\ e''_k\}$. Let the result be E_3. We can do this shuffling because the y are never used anywhere.

Step 4. Let $E_4 = A\ minus\ (B_1 \cup ... \cup B_k)$ where $A, B_1, ..., B_k$ are obtained from E_3 as below. It should be obvious that E_4 and E_3 are equivalent.

- $A = \{e \mid x_1 \in proj\ z, ..., x_n \in proj\ z, y'_1 \in eq_b\ e'_1, ..., y'_m \in eq_b\ e'_m\}$
- $B_i = \{e \mid x_1 \in proj\ z, ..., x_n \in proj\ z, y''_i \in e''_i\}$

At this point, B_i are smaller expressions that have the same form as E, the expression we started off with. (Well, actually some of the e''_i may be of the form $e_{left} \cup e_{right}$. But the $\cup$ can be pulled out to yield the right form.) So the above steps can be repeated to progressively replace the rest of the *not* by *minus*. After that the only form of set abstractions we have to deal with is that of A.

In A, since we started off with normalised expression of height 1, it is the case that e, e'_1, ..., e'_m are all height 0. So they must be built entirely out of projections, tupling, $()$, and variables (x_1, ..., x_n) of height 0. It is easy to see that A can be expressed in $\mathcal{TRA}$ by an expression of the form $M_1 \circ M_2 \circ M_3$ where M_3 is a chain of cartesian products, M_2 is a chain of selections, and M_1 is a chain of projections. □

Therefore, we conclude that queries definable in flat relational algebra are exactly those definable functions of relative set abstraction which have flat relational query types. This connection is extremely tight.

Paredaens and Van Gucht [18, 17] proved a similar conservative extension result for the nested relational algebra of Thomas and Fischer [20]. The Thomas and Fischer algebra is very restrictive and its operators

can be applied only to the topmost level of nested relations. Therefore our result in this section can be regarded as an extension of Paredaens and Van Gucht's.

The key to the proof of the Conservative Extension Theorem is the Normal Form Theorem. The heart of Paredeans and Van Gucht's proof is also a similar normal form result. However, their normal form result is a normal form of logic formula and the intuition behind their proof is mainly that of logical equivalence. In our case, our inspiration comes from a well known optimisation strategy (and we cannot resist citing Wadler's early paper [25, 26], which has a very amusing title, on this subject). In plain terms, we have evaluated the query without looking at the input and managed to flatten the query sufficiently until all intermediate operators of higher heights are "optimised out." This idea is succintly summarised by the rewriting rule $\mathcal{C}[\{e \mid \Delta_1, x \in \{e' \mid \Delta'\}, \Delta_2\}] \leadsto \mathcal{C}[\{e[e'/x] \mid \Delta_1, \Delta', \Delta_2[e'/x]\}]$ which eliminates the intermediate set built by $\{e' \mid \Delta'\}$.

5 Some easy consequences of these results

In the course of proving our main result, we have shown that expressions of relative set abstraction can be reduced to very simple normal forms. Normal forms can be exploited in many proofs of undefinability by showing that there is no normal form that defines the desired function. An ripe example of this sort is the undefinability of conditional on base type.

5.1 Corollary

A conditional on base type is a function $cond : \{unit\} \times b \times b \to b$ *such that* $cond(c_0, (c_1, c_2))$ *is equal to* c_1 *if* c_0 *is equal to* $\{()\}$ *and is equal to* c_2 *if* c_0 *is equal to* $\{\}$*. It is not definable in* $\mathcal{RSA}$*,* $\mathcal{NRA}$*, and* $\mathcal{NRC}$*.*

Proof. It suffices to check that there is no $\mathcal{RSA}$ normal form that defines $cond : \{unit\} \times b \times b \to b$. Suppose to the contrary that it is definable. Then it is definable by a closed normal form $\lambda x.e$. But e cannot be any of the following: $()$, $\{\}$, (e_1, e_2), $\{e' \mid \Delta\}$, $e_1 \cup e_2$, $not\ e'$, $eq_b\ e'$, $\pi_1\ x$, or $\pi_2\ x$, because they do not have type b. It cannot be any of the following either: c of type b, $\pi_2(\pi_1\ x)$, or $\pi_2(\pi_2\ x)$, because they are clearly not the conditional. As there are no other alternatives, we have arrived at a contradiction. So conditional on base type is not definable in our languages. □

Here is another example of this nature that illustrates the fact that there is no way to "get out of sets" in our languages.

5.2 Corollary

Every definable function of type $\{s\} \to b$ *must be a constant function.*

Proof. We show that all normal forms of type $\{s\} \to b$ without free variables are constant functions. Let $\lambda x.e : \{s\} \to b$ be a normal form. Suppose e is not the constant c or $()$. Then e must look like (e, e), $e \cup e$, $\{\}$, $\{e \mid \Delta\}$, $not(e)$, $eq_b(e)$, or is a projection on the variable x. But all of these alternatives

are badly typed. So $\lambda x.e$ is a constant function. □

There are a number of well known theorems in flat relational query languages. The tight relationship we have demonstrated between our query language and the flat relational algebra enables us to draw a few (otherwise not so obvious) conclusions. An example of this is the undefinability of transitive closure.

5.3 Corollary

Transitive closure is not expressible in $\mathcal{RSA}$, $\mathcal{NRC}$, and $\mathcal{NRA}$.

Proof. Suppose it is expressible. Then we can express transitive closure of a binary relation on a base type (such as integer). Since this is a function from a flat relation to a flat relation, by the Conservative-extension theorem, it is expressible in flat relational algebra. This contradicts the well known result on flat relational algebra (see Aho and Ullman [4] or Maier [15]). Hence it is not definable in our languages. □

It was worked out in a previous paper [7] that the language of Abiteboul and Beeri [2] is obtained by adding the powerset operator to $\mathcal{NRC}$. If the purpose of this addition is to increase expressive power, it is unlikely to be practical. The reason is that practically all the interesting new queries that can now be written must involve the expensive powerset operation. Transitive closure is an example that immediately comes to mind.

5.4 Corollary

Transitive closure can only be expressed in the language of Abiteboul and Beeri via an excursion through the powerset. □

6 Two extensions to the main theorem

In this final section, we extend $\mathcal{NRC}$ to $\mathcal{NRC}+$ by a variant type mechanism. Then we provide a proof that this extended language is conservative with respect to set height. Furthermore, the proof holds uniformly when the language is interpreted under a set-, list-, or bag-based semantics.

Types. Variant types are added to the language. If s and t are object types, then the variant type $s + t$ is also an object type. Basically, the domain of a variant type $s + t$ is the union of the domains of s and t but values from s are tagged with a 1-tag and values from t are tagged with a 2-tag.

Expressions. Three new constructs are required to manipulate variant objects. Their formation rules are listed below:

$$\frac{e : s}{\mathit{left}^t\ e : s + t} \qquad \frac{e : s}{\mathit{right}^t\ e : t + s} \qquad \frac{e_1 : s_1 + s_2 \qquad e_2 : t \qquad e_3 : t}{\mathit{case}\ e_1\ \mathit{of}\ \mathit{left}\ x^{s_1} \Rightarrow e_2 \mid \mathit{right}\ x^{s_2} \Rightarrow e_3 : t}$$

Semantics. We offer an informal explanation. *left* e injects e into a variant object by tagging the object denoted by e with a 1-tag. *right* e injects e into a variant object by tagging the object denoted by e with a 2-tag. $\mathit{case}\ e_1\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ e_2 \mid \mathit{right}\ y\ \Rightarrow\ e_3$ processes the variant object denoted by e_1 as follows. If e_1 is equal to *left* e, then the case expression is equal to $e_2[e/x]$. If e_1 is equal to *right* e, then the case expression is equal to $e_3[e/x]$. That is, the left or the right branch is taken depending on whether e_1 has a 1-tag or a 2-tag respectively.

Example. $\bigcup\{(\mathit{case}\ x\ \mathit{of}\ \mathit{left}\ y\ \Rightarrow\ \{y\} \mid \mathit{right}\ z\ \Rightarrow\ \{\}) \mid x \in X\}$ denotes the selection of items that are 1-tagged in the set X.

Let us now proceed with our last result.

6.1 Theorem

Let $e : s$ be an expression of $\mathcal{NRC}+$. Then there is an equivalent expression e' such that $ht(e') \leq$ $\max(\{ht(s)\} \cup \{ht(s) \mid s$ is the object type of a free variable in $e\})$.

Proof. Consider the following rewriting systems.

1. $\mathcal{C}[(\lambda x.e)e'] \leadsto \mathcal{C}[e[e'/x]]$
2. $\mathcal{C}[\pi_1(e_1, e_2)] \leadsto \mathcal{C}[e_1]$
3. $\mathcal{C}[\pi_2(e_1, e_2)] \leadsto \mathcal{C}[e_2]$
4. $\mathcal{C}[e\ (\mathit{case}\ e_1\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ e_2 \mid \mathit{right}\ y\ \Rightarrow\ e_3)] \leadsto \mathcal{C}[\mathit{case}\ e_1\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ e\ e_2 \mid \mathit{right}\ y\ \Rightarrow\ e\ e_3]$
5. $\mathcal{C}[\pi_i\ (\mathit{case}\ e_1\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ e_2 \mid \mathit{right}\ y\ \Rightarrow\ e_3)] \leadsto \mathcal{C}[\mathit{case}\ e_1\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ \pi_i\ e_2 \mid \mathit{right}\ y\ \Rightarrow\ \pi_i\ e_3]$
6. $\mathcal{C}[\mathit{case}\ \mathit{left}\ e\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ e_2 \mid \mathit{right}\ y\ \Rightarrow\ e_3] \leadsto \mathcal{C}[e_2[e/x]]$
7. $\mathcal{C}[\mathit{case}\ \mathit{right}\ e\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ e_2 \mid \mathit{right}\ y\ \Rightarrow\ e_3] \leadsto \mathcal{C}[e_3[e/x]]$
8. $\mathcal{C}[\mathit{case}\ (\mathit{case}\ e_1'\ \mathit{of}\ \mathit{left}\ x'\ \Rightarrow\ e_2' \mid \mathit{right}\ y'\ \Rightarrow\ e_3')\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ e_2 \mid \mathit{right}\ y\ \Rightarrow\ e_3] \leadsto$ $\mathcal{C}[\mathit{case}\ e_1'\ \mathit{of}\ \mathit{left}\ x'\ \Rightarrow\ (\mathit{case}\ e_2'\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow\ e_2 \mid \mathit{right}\ y\ \Rightarrow e_3) \mid \mathit{right}\ y'\ \Rightarrow (\mathit{case}\ e_3'\ \mathit{of}\ \mathit{left}\ x\ \Rightarrow$ $e_2 \mid \mathit{right}\ y\ \Rightarrow e_3)]$
9. $\mathcal{C}[\bigcup\{e \mid x \in (e_1 \cup e_2)\}] \leadsto \mathcal{C}[(\bigcup\{e \mid x \in e_1\}) \cup (\bigcup\{e \mid x \in e_2\})]$
10. $\mathcal{C}[\bigcup\{e \mid x \in \{\}\}] \leadsto \mathcal{C}[\{\}]$
11. $\mathcal{C}[\bigcup\{e \mid x \in \{e'\}\}] \leadsto \mathcal{C}[e[e'/x]]$
12. $\mathcal{C}[\bigcup\{e_1 \mid x_1 \in \bigcup\{e_2 \mid x_2 \in e_3\}\}] \leadsto \mathcal{C}[\bigcup\{\bigcup\{e_1 \mid x_1 \in e_2\} \mid x_2 \in e_3\}]$
13. $\mathcal{C}[\bigcup\{e_1 \mid x_1 \in (\mathit{case}\ e_2\ \mathit{of}\ \mathit{left}\ x_2\ \Rightarrow\ e_3 \mid \mathit{right}\ x_3\ \Rightarrow\ e_4)\}] \leadsto \mathcal{C}[\mathit{case}\ e_2\ \mathit{of}\ \mathit{left}\ x_2\ \Rightarrow\ \bigcup\{e_1 \mid x_1 \in$ $e_3\} \mid \mathit{right}\ x_3\ \Rightarrow\ \bigcup\{e_1 \mid x_1\ \in e_4\}]$

Let $k = \max\{ht(t) \mid t$ is the object type of a free variable in $e\}$. Suppose e has a normal form e' under the above rewriting rules. We show by structural induction on e' that e' satisfies the requirement of the theorem.

Case $e' : s$ is a chain of projections on a variable, $()$, $\{\}$, *not*, eq_b, or c. Immediate.

Case $e' : s$ is $\lambda x.e_1$ where $x : s_1$ and $e_1 : s_2$. Then s is $s_1 \to s_2$. We have $ht(e_1) \leq \max(k, ht(s_2), ht(s_1))$ by hypothesis. But $ht(e') = \max(ht(s_1), ht(e_1))$. So $ht(e') \leq \max(k, ht(s))$.

Case $e' : s$ is (e_1, e_2) where $e_1 : s_1$ and $e_2 : s_2$. Then s is $s_1 \times s_2$. By hypothesis, $ht(e_1) \leq \max(k, ht(s_1))$ and $ht(e_2) \leq \max(k, ht(s_2))$. So $ht(e') = \max(ht(e_1), ht(e_2)) \leq \max(k, ht(s))$.

Case $e' : s$ is *not* e_1. Then s is $\{unit\}$ and $e_1 : \{unit\}$. Then $ht(e') = ht(e_1)$. The case holds by hypothesis on e_1.

Case $e' : s$ is $eq_b\ e_1$. Then s is $\{unit\}$ and $e_1 : b \times b$. Then $ht(e') = \max(ht(s), ht(e_1))$. By hypothesis, $ht(e_1) \leq k$. The case holds.

Case $e' : s$ is $e_1 \cup e_2$. Then $ht(e') = \max(ht(e_1), ht(e_2))$. By hypothesis, $ht(e_1) \leq \max(k, ht(s))$ and $ht(e_2) \leq \max(k, ht(s))$. Therefore, $ht(e') \leq \max(k, ht(s))$.

Case $e' : s$ is $\{e_1\}$ where $e_1 : s_1$. Then s is $\{s_1\}$. By hypothesis, $ht(e_1) \leq \max(k, ht(s_1))$. So $ht(e') = \max(ht(e_1), ht(s)) \leq \max(k, ht(s))$.

Case $e' : s$ is $\bigcup\{e_1 \mid x \in e_2\}$ where $e_2 : \{s_2\}$. Because e' is a normal form under rules 1 to 12, e_2 must be a chain of projections on a variable or has the form *not* e_3 or $eq_b\ e_3$. Hence $ht(e_2) \leq \max(k, 1)$. So $ht(x) = ht(e_2) - 1 \leq k$. Then, by hypothesis, $ht(e_1) \leq \max(k, ht(x), ht(s)) = \max(k, ht(s))$. Then $ht(e') = \max(ht(s), ht(e_1), ht(e_2)) \leq \max(k, ht(s))$.

Case $e' : s$ is $left^{s_2}\ e_1$ where $e_1 : s_1$. Then s is $s_1 + s_2$. By hypothesis, $ht(e_1) \leq \max(k, ht(s_1))$. So $ht(e') = \max(ht(e_1), ht(s)) \leq \max(k, ht(s))$.

Case $e' : s$ is *right* e_1. Similar to previous case.

Case $e' : s$ is *case* e_1 *of left* $x \Rightarrow e_2 \mid$ *right* $y \Rightarrow e_3$ where $e_1 : s_1 + s_2$. Then $x : s_1$, $y : s_2$, $e_2 : s$, and $e_3 : s$. Since $e' : s$ is a normal form under rules 1 to 12, e_1 must be a chain of projections on a free variable. Hence $ht(e_1) \leq k$. Consequently, $ht(s_1) \leq k$ and $ht(s_2) \leq k$. By hypothesis, $ht(e_2) \leq \max(k, ht(x), ht(s)) = \max(k, ht(s))$. Similarly, $ht(e_3) \leq \max(k, ht(y), ht(s)) = \max(k, ht(s))$. Now $ht(e') = \max(ht(e_1), ht(e_2), ht(e_3)) \leq \max(k, ht(s))$.

Finally, we have to show that the normal form e' of e exists. To do this, we prove that the rewriting system is strongly normalising. Let φ maps variable names to natural numbers greater than 1. Let $\varphi[n/x]$ be the function that maps x to n and agrees with φ on other variables. Let $\|e\|\varphi$, defined below, measure the size of e in the environment φ where each free variable x in e is given the size $\varphi(x)$.

- $\|x\|\varphi = \varphi(x)$
- $\|c\|\varphi = \|()\|\varphi = \|\{\}\|\varphi = \|not\|\varphi = \|eq_b\|\varphi = 2$

- $\|\pi_1\ e\|\varphi = \|\pi_2\ e\|\varphi = \|not\ e\|\varphi = \|eq_b\ e\|\varphi = \|left\ e\|\varphi = \|right\ e\|\varphi = \|\{e\}\|\varphi = 2 \cdot \|e\|\varphi$
- $\|\lambda x.e\|\varphi = \|e\|\varphi[2/x]$
- $\|(\lambda x.e)(e')\|\varphi = \|e\|\varphi[\|e'\|\varphi/x] \cdot \|e'\|\varphi$
- $\|(e_1, e_2)\|\varphi = \|e_1\|\varphi + \|e_2\|\varphi$
- $\|e_1 \ \cup \ e_2\|\varphi = 1 + \|e_1\|\varphi + \|e_2\|\varphi$
- $\|\bigcup\{e' \mid x \in e\}\|\varphi = (\|e'\|\varphi[\|e\|\varphi/x] + 1) \cdot \|e\|\varphi$
- $\|case\ e_1\ of\ left\ x\ \Rightarrow\ e_2\ of\ right\ y\ \Rightarrow e_3\|\varphi = \|e_1\|\varphi \cdot (1 + \|e_2\|\varphi[\|e_1\|\varphi/x] + \|e_3\|\varphi[\|e_1\|\varphi/y])$

A lemma. Let φ_1 and φ_2 be such that for every x, $\varphi_1(x) \leq \varphi_2(x)$. By a routine induction on e, we have $\|e\|\varphi_1 \leq \|e\|\varphi_2$. **A corollary of the lemma.** Let $\|e'\|\varphi \leq n$. By an induction on e and the previous lemma, we have $\|e[e'/x]\|\varphi \leq \|e\|\varphi[n/x]$. Then it is readily verified that whenever $e \leadsto e'$, we have $\|e\|\varphi > \|e'\|\varphi$ for any choice of φ. Therefore, the rewriting system is strongly normalizing. This completes the proof. □

As remarked earlier in the report, variant mechanisms have been used in some data models such as [3] and [13]. However, many earlier interesting works on expressive power omitted them from considerations [12, 8, 2]. We hope the above result have rectified this situation to some extent.

Our languages have been given semantics based on sets. These languages can be given semantics based on bags or on lists. For example, $\mathcal{NRC}$ can be treated as a "nested bag calculus" by interpreting $\{\}$ as the empty bag, $e_1 \cup e_2$ as union of bags, and $\bigcup\{e' \mid x \in e\}$ as flatmapping the function $\lambda x.e'$ over the bag e. Similarly, $\mathcal{NRC}$ can be treated as a "nested list calculus" by treating $\{\}$ as the empty list, $e_1 \cup e_2$ as the concatenation of list e_1 to the list e_2, and $\bigcup\{e' \mid x \in e\}$ as flatmapping the function $\lambda x.e'$ over the list e. It is easy to check that the rewriting rules given in this section are valid for bag semantics as well as for list semantics. So the same proof above works for "nested bag calculus" and for "nested list calculus." In fact, it works even in the presence of variant types.

The uniformity of this proof allows us to draw a few useful conclusions. It has been observed earlier that the translations between $\mathcal{RSA}$ and $\mathcal{NRC}$ preserve set height. Therefore, the General Conservative Extension Theorem holds also for "relative bag abstraction" and for "relative list abstraction." Similarly, it follows that it holds also for "nested bag algebra" and for "nested list algebra." [It must be remarked that these conclusions cannot be reached from the proof given in Section 3. The proof in Section 3 does not work when $\mathcal{RSA}$ is interpreted using a list semantics. This is because one of the rules used in Section 3 (namely rule 1) is not valid as list concatenation does not commute.]

Naturally $\mathcal{RSA}$ can be extended with exactly the same variant type constructs presented in this section without affecting the theorem on general conservative extension. Although no detail of $\mathcal{NRA}$ is given in this report, it is worth mentioning the extension of $\mathcal{NRA}$ with variant mechanism as it is quite interesting. The expected coproduct constructs $left^{s,t} : s \to s + t$, $right^{s,t} : t \to s + t$, and $(f|g) : s + s' \to t$ where $f : s \to t$ and $g : s' \to t$ must be added. In addition, to retain the simplicity of the translations between $\mathcal{NRC}$ and $\mathcal{NRA}$ given by [7], we must also add an operator $\delta^{s,t,t'} : s \times (t + t') \to (s \times t) + (s \times t')$. The operator δ basically pushes the left and right injections over a product. That

is, it satisfies $\delta \circ \langle f, \mathit{left}\rangle = \mathit{left} \circ \langle f, \mathit{id}\rangle$ and $\delta \circ \langle f, \mathit{right}\rangle = \mathit{right} \circ \langle f, \mathit{id}\rangle$. It is then a straight forward exercise to maintain height preserving translations between $\mathcal{NRA}+$ and $\mathcal{NRC}+$.

Acknowledgements. The author thanks Val Breazu-Tannen and Peter Buneman for many useful discussions and invaluable suggestions, and Dirk Van Gucht for explaining a fine point of his conservativeness result. The author is grateful to the National Science Foundation and the Army Research Office for financial support.

References

[1] S. Abiteboul, C. Beeri, M. Gyssens, and D. Van Gucht. An Introduction to the Completeness of Languages for Complex Objects and Nested Relations. In S. Abiteboul, P. C. Fisher, and H.-J. Schek, editors, *LNCS 361: Nested Relations and Complex Objects in Databases*, pages 117–138. Springer-Verlag, 1987.

[2] Serge Abiteboul and Catriel Beeri. On the Power of Languages for the Manipulation of Complex Objects. In *Proceedings of International Workshop on Theory and Applications of Nested Relations and Complex Objects*, Darmstadt, 1988.

[3] Serge Abiteboul and Richard Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[4] Alfred V. Aho and Jeffrey D. Ullman. Universality of Data Retrieval Languages. In *Prceedings 6th POPL, Texas, January 1979*, pages 110–120, 1979.

[5] V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural Recursion as a Query Language. In *Proceedings of 3rd International Workshop on Database Programming Languages*, pages 9–19, Naphlion, Greece, August 1991. Morgan Kaufmann.

[6] V. Breazu-Tannen and R. Subrahmanyam. Logical and Computational Aspects of Programming with Sets/Bags/Lists. In *LNCS 510: Proceedings of 18th International Colloquium on Automata, Languages, and Programming, Madrid, Spain, July 1991*, pages 60–75. Springer Verlag, 1991.

[7] Val Breazu-Tannen, Peter Buneman, and Limsoon Wong. Natually Embedded Query Languages. In *LNCS ?: Proceedings of International Conference on Database Theory, Berlin, Germany, October, 1992*. Springer-Verlag, To appear.

[8] Stephane Grumbach and Victor Vianu. Playing Games with Objects. In S. Abiteboul and P. C. Kanellakis, editors, *LNCS 470: 3rd International Conference on Database Theory, Paris, France, December 1990*, pages 25–39. Springer-Verlag, 1990.

[9] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press, 1992.

[10] Marc Gyssens and Dirk Van Gucht. A Comparison Between Algebraic Query Languages for Flat and Nested Databases. *Theoretical Computer Science*, 87:263–286, 1991.

[11] P. Hudak and P. Wadler. Report on the Programming Language Haskell. Technical Report 90/?, Glasgow University, Glasgow G12 8QQ, Scotland, April 1990.

[12] Richard Hull and Jianwen Su. On the Expressive Power of Database Queries with Intermediate Types. *Journal of Computer and System Sciences*, 43:219–267, 1991.

[13] Richard Hull and Chee K. Yap. The Format Model: A Theory of Database Organisation. *Journal of the ACM*, 31(3):518–537, July 1984.

[14] G. Jaeschke and H. J. Schek. Remarks on the Algebra of Nonfirst Normal Form Relations. In *Proceedings ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, Los Angeles, California, March 1982.

[15] David Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville, Maryland, 1983.

[16] Eugenio Moggi. Notions of Computation and Monads. *Information and Computation*, 93:55–92, 1991.

[17] Jan Paredaens and Dirk Van Gucht. Possibilities and Limitations of Using Flat Operators in Nested Algebra Expressions. In *Proceedings of 7th ACM Symposium on Principles of Database Systems,Austin, Texas*, pages 29–38, 1988.

[18] Jan Paredaens and Dirk Van Gucht. Converting Nested Relational Algebra Expressions into Flat Algebra Expressions. *ACM Transaction on Database Systems*, 17(1):65–93, March 1992.

[19] David A. Schmidt. *Denotational Semantics: A Methodology For Language Development.* Allyn and Bacon, Inc., Boston, 1986.

[20] S. J. Thomas and P. C. Fischer. Nested Relational Structures. In P. C. Kanellakis, editor, *Advances in Computing Research: The Theory of Databases*, pages 269–307. JAI Press, 1986.

[21] P. W. Trinder. Comprehension: A Query Notation for DBPLs. In *Proceedings of 3rd International Workshop on Database Programming Languages*, pages 49–62, Nahplion, Greece, August 1990. Morgan Kaufmann. In press.

[22] P. W. Trinder and P. L. Wadler. List Comprehensions and the Relational Calculus. In *Proceedings of 1988 Glasgow Workshop on Functional Programming*, pages 115–123, Rothesay, Scottland, August 1988.

[23] David Turner. Recursion Equations as a Programming Language. In J. Darlington, P. Henderson, and David Turner, editors, *Functional Programming and its Applications.* Cambridge University Press, 1982.

[24] David Turner. Miranda—a Non-strict Functional Language with Polymorphic Types. In *LNCS 201: Proceedings of Conference on Functional Programming Languages and Computer Architecture, Nancy, 1985*, pages 1–16. Springer-Verlag, 1985.

[25] Philip Wadler. Listlessness is better than laziness. In *Proceedings of ACM Symposium on Lisp and Functional Programming*, Austin, Texas, August 1984.

[26] Philip Wadler. Listlessness is better than laziness II. In H. Ganzinger and N. D. Jones, editors, *LNCS 217: Programs as Data Objects.* Springer-Verlag, October 1985.

[27] Philip Wadler. Comprehending Monads. In *Proceedings of ACM Conference on Lisp and Functional Programming*, Nice, June 1990.

[28] David A. Watt and Phil Trinder. Towards a Theory of Bulk Types. Fide Technical Report 91/26, Glasgow University, Glasgow G12 8QQ, Scotland, July 1991.