Conservativity of Nested Relational Calculi with Internal Generic Functions*

Leonid Libkin and Limsoon Wong

Department of Computer and Information Science,
University of Pennsylvania,
Philadelphia, PA 19104-6389, USA.
email: {libkin, limsoon}@saul.cis.upenn.edu

Abstract

It is known that queries in nested relational calculus are independent of the depth of set nesting in the intermediate data and this remains true in the presence of aggregate functions. We prove that this continues to be true if the calculus is augmented with any internal generic family of functions.

1 Introduction

Paredaens and Van Gucht [7] proved that the nested relational calculus is no more expressive than the traditional relational calculus when input and output are flat relations. That is, every query on input and output whose depth of set nesting is at most 1 (or flat relations) can be expressed without using any intermediate data whose depth of set nesting is greater than 1. This result was generalized by Wong [10] who showed that every query on input and output whose depth of set nesting is at most k can be expressed without using any intermediate data whose depth of set nesting is greater than k. Hence the expressive power of the nested relational calculus is *independent* of the depth of set nesting allowed in the intermediate data. This property is called the *conservative extension property.* Libkin and Wong [5] showed that when the nested relational calculus is endowed with aggregate functions, it retains the conservative extension property.

The aim of this paper is to demonstrate that the nested relational calculus continues to retain the conservative extension property even in the presence of any family of functions that are generic and do not invent new values. The strategy is to combine the rewriting technique of Wong [10] and the effective lifting of linear orders of Libkin and Wong [5] to encode any such function into one whose height is minimum.

*Appeared in *Information Processing Letters* **49**(6):273–280, March 1994.

Lambda Calculus and Products

$$\frac{}{x^s : s} \qquad \frac{e : t}{\lambda x^s.e : s \to t} \qquad \frac{e_1 : s \to t \quad e_2 : s}{e_1\ e_2 : t} \qquad \frac{}{() : unit} \qquad \frac{e : s \times t}{\pi_1\ e : s \quad \pi_2\ e : t} \qquad \frac{e_1 : s \quad e_2 : t}{(e_1, e_2) : s \times t}$$

Set Monad

$$\frac{}{\{\}^s : \{s\}} \qquad \frac{e : s}{\{e\} : \{s\}} \qquad \frac{e_1 : \{s\} \quad e_2 : \{s\}}{e_1 \cup e_2 : \{s\}} \qquad \frac{e_1 : \{t\} \quad e_2 : \{s\}}{\bigcup\{e_1 \mid x^s \in e_2\} : \{t\}}$$

Booleans

$$\frac{}{true : \mathbb{B}} \qquad \frac{}{false : \mathbb{B}} \qquad \frac{e_1 : \mathbb{B} \quad e_2 : t \quad e_3 : t}{if\ e_1\ then\ e_2\ else\ e_3 : t}$$

2 Nested relational calculus

We use the nested relational calculus $\mathcal{NRC}$ as presented in Breazu-Tannen, Buneman, and Wong [3]. In this section, it is extended with natural numbers, simple arithmetics, and a summation operator.

Types. A type in $\mathcal{NRC}$ is either a complex object type or is a function type $s \to t$ where s and t are complex object types. The complex object types are given by the grammar:

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

Objects of type $\mathbb{B}$ are the two boolean values *true* and *false*. The unique object of type *unit* is denoted by $()$. Objects of type $s \times t$ are pairs whose first (or left) components are objects of type s and second (or right) components are objects of type t. Objects of type $\{s\}$ are finite sets of objects of type s. We also included some uninterpreted base types b.

Expressions. Expressions of $\mathcal{NRC}$ are constructed using the rules in the figure. The language also contains some uninterpreted constants c of base type $Type(c)$ and uninterpreted functions p of function type $Type(p)$.

The type superscripts are omitted in the rest of the paper because they can be inferred [6]. The semantics of $\mathcal{NRC}$ was described in [3]. We repeat the meaning of the set monad constructs here. $\{\}$ is the empty set. $\{e\}$ is the singleton set containing e. $e_1 \cup e_2$ is the union of sets e_1 and e_2. The construct $\bigcup\{e_1 \mid x \in e_2\}$ denotes the set obtained by first applying the function $\lambda x.e_1$ to elements of the set e_2 and then taking their union. The shorthand $\{o_1, \ldots. o_n\}$ is used to denote $\{o_1\} \cup \ldots \cup \{o_n\}$, provided o_1, ..., o_n are distinct objects.

As it stands, $\mathcal{NRC}$ can merely express queries that are purely structural. It was shown in [3] that endowing $\mathcal{NRC}$ with equality test $=^s$: $s \times s \to \mathbb{B}$ at all types s elevates $\mathcal{NRC}$ to a fully fledged nested relational language (in fact, equivalent to classical nested relational algebra of [8]). That is, operations such as nest, membership test $\in^s$: $s \times \{s\} \to \mathbb{B}$, subset test $\subseteq^s$: $\{s\} \times \{s\} \to \mathbb{B}$, set intersection, set difference, etc. are expressible in $\mathcal{NRC}(=)$ (we write the additional primitive in brackets to distinguish various extensions of the language).

Practical database query languages frequently have to deal with queries such as "select maximum of column," "select count from column," etc. To handle this kind of queries, we add natural numbers (whose type is denoted by $\mathbb{N}$) and the following constructs:

$$\frac{e_1 : \mathbb{N} \quad e_2 : \mathbb{N}}{e_1 + e_2 : \mathbb{N}} \qquad \frac{e_1 : \mathbb{N} \quad e_2 : \mathbb{N}}{e_1 \cdot e_2 : \mathbb{N}} \qquad \frac{e_1 : \mathbb{N} \quad e_2 : \mathbb{N}}{e_1 \dot{-} e_2 : \mathbb{N}} \qquad \frac{e_1 : \mathbb{N} \quad e_2 : \{s\}}{\sum\{e_1 \mid x^s \in e_2\} : \mathbb{N}}$$

where $+$, $\cdot$, and $\dot{-}$ are respectively addition, multiplication, and modified subtraction on natural numbers. (That is, if $n \leq m$, then $n \dot{-} m = 0$; if $n > m$, then $n \dot{-} m = n - m$.) The summation construct $\sum\{e_1 \mid x^s \in e_2\}$ denotes the number obtained by first applying the function $\lambda x.e_1$ to every item in the set e_2 and then adding the results up. That is, $\sum\{e_1 \mid x \in X\}$ is $f(o_1) + \ldots + f(o_n)$ if f is the function denoted by $\lambda x.e_1$ and $\{o_1, \ldots o_n\}$ is the set denoted by X. It should be stressed that in the construct $\sum\{e_1 \mid x \in e_2\}$, the $\{e_1 \mid x \in e_2\}$ part is not an expression; hence this construct is stronger than just adding $\sum : \{\mathbb{N}\} \to \mathbb{N}$.

The extended language $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =)$, where the additional base types and primitives are explicitly listed between brackets, is capable of expressing many aggregate functions found in commercial databases. Here are two examples: "count the number of records in R" is $count(R) \triangleq \sum\{1 \mid x \in R\}$ and "total up the first column of R" is $total(R) \triangleq \sum\{\pi_1\ x \mid x \in R\}$.

Now we formally define the conservative extension property. The set height $ht(s)$ of a type s is defined by induction on the structure of type: $ht(unit) = ht(b) = 0$, $ht(s \times t) = ht(s \to t) = \max(ht(s), ht(t))$, and $ht(\{s\}) = 1 + ht(s)$. Every expression of our language has a unique typing derivation. The set height of expression e is defined as $ht(e) = \max\{ht(s) \mid s$ occurs in the type derivation of $e\}$. Let $\mathcal{L}_{i,o,h}$ denote the class of functions whose input has set height at most i, whose output has set height at most o, and which are definable in the language $\mathcal{L}$ using an expression whose set height is at most $h \geq \max(i, o)$. $\mathcal{L}$ is said to have the *conservative extension property with fixed constant* k if $\mathcal{L}_{i,o,h} = \mathcal{L}_{i,o,h+1}$ for all i, o, and $h \geq \max(i, o, k)$.

It is known from Wong [10] and Libkin and Wong [5] that

Theorem 2.1 $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =)$ *has the conservative extension property with fixed constant* 0. *Moreover,* $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =)$ *endowed with any additional primitive* $p : s \to t$ *also has the conservative extension property with fixed constant* $ht(s \to t)$. □

3 Lifting of linear orders

In this section we present a technique to lift linear order from base types to all types. This technique plays the key role in the encoding used in the next section.

The following lemma is a folklore (see Wechler [9]):

Lemma 3.1 *Given a partially ordered set $\langle A, \leq \rangle$, define an ordering $\precsim$ on its finite powerset $\mathcal{P}_{\mathrm{fin}}(A)$ as follows: $X \precsim Y$ iff $\max((X-Y) \cup (Y-X)) \subseteq Y$, or, equivalently, if $\forall x \in X-Y \exists y \in Y-X : x \leq y$. Then $\precsim$ is a partial order. Moreover, if $\leq$ is linear, then so is $\precsim$.* □

This way of lifting linear order can be easily expressed in $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =)$. Therefore, we have

Theorem 3.2 *Suppose a linear order $\leq^b$ is given for each base type b. Then a linear order $\leq^t$ is computable by $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =)$ for each type t.*

Proof. Define $\leq^t$ by induction on types. For a base type $\leq^b$ is given. For pairs it is defined lexicographically: $x \leq^{t \times s} y \triangleq \mathit{if}\ \pi_1\ x \leq^s \pi_1\ y\ \mathit{then}\ (\mathit{if}\ \pi_1\ x =^s \pi_1\ y\ \mathit{then}\ \pi_2\ x \leq^t \pi_2\ y\ \mathit{else\ true})\ \mathit{else\ false}$.

For set types we use lemma 3.1: $X \leq^{\{s\}} Y \triangleq \mathit{if}\ X \sqsubseteq^\flat_s Y\ \mathit{then}\ (\mathit{if}\ Y \sqsubseteq^\flat_s X\ \mathit{then}\ X \leq^\flat_s Y\ \mathit{else\ true})\ \mathit{else\ false}$. Here $X \sqsubseteq^\flat_s Y$ iff $\forall x \in X \exists y \in Y :\ x \leq^s y$ and $X \leq^\flat_s Y$ iff $X - Y \sqsubseteq^\flat_s Y - X$. These can be implemented as follows. $X \sqsubseteq^\flat_s Y \triangleq 0 =^{\mathbb{B}} \Sigma\{\mathit{if}\ (\Sigma\{\mathit{if}\ x \leq^s y\ \mathit{then}\ 1\ \mathit{else}\ 0 \mid y \in Y\}) =^{\mathbb{N}} 0\ \mathit{then}\ 1\ \mathit{else}\ 0 \mid x \in X\}$ and $X \leq^\flat_s Y \triangleq (\Sigma\{\mathit{if}\ x \in^s Y\ \mathit{then}\ 0\ \mathit{else}\ (\mathit{if}\ (\Sigma\{\mathit{if}\ y \in^s X\ \mathit{then}\ 0\ \mathit{else}\ (\mathit{if}\ x \leq^s y\ \mathit{then}\ 1\ \mathit{else}\ 0) \mid y \in Y\}) =^{\mathbb{N}} 0\ \mathit{then}\ 1\ \mathit{else}\ 0) \mid x \in X\}) =^{\mathbb{N}} 0$. □

We denote $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =)$ endowed with linear orders at base types by $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$. Notice that if $\mathbb{B}$ and $\mathbb{N}$ are the only base types, then this does not add expressive power since the orderings on booleans and naturals are definable: $\mathit{false} \leq^{\mathbb{B}} \mathit{true}$ and $n \leq^{\mathbb{N}} m \triangleq (n \dot{-} m =^{\mathbb{N}} 0)$.

Corollary 3.3 *A rank assignment is a function $\mathit{sort} : \{s\} \to \{s \times \mathbb{N}\}$ such that $\mathit{sort}\{a_1, \ldots, a_n\} = \{(a_1, 1), \ldots, (a_n, n)\}$ where $a_1 \precnsim^s \ldots \precnsim^s a_n$. Rank assignment is expressible in $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$.*

Proof. Define $\mathit{sort}(x) \triangleq \bigcup\{(r, \Sigma\{\mathit{if}\ c \leq^s r\ \mathit{then}\ 1\ \mathit{else}\ 0 \mid c \in x\}) \mid r \in x\}$. □

Finally, we have the following

Corollary 3.4 *If all orderings on the base types are well-founded, then so are the lifted orderings. Furthermore, if constant $\mathit{min}^b : b$ and function $\mathit{succ}^b : b \to b$ (meaning minimal element and successor) are given for each base type b, they can be defined for any type.*

Proof. The first statement follows from the fact that $X \precsim Y$ implies $X \sqsubseteq^\flat Y$ where $X \sqsubseteq^\flat Y$ iff $\forall x \in X \exists y \in Y : x \leq y$, and $\sqsubseteq^\flat$ is known to be well-founded if $\leq$ is [2]. Minimal elements are definable as follows: $\mathit{min}^{t \times s} = (\mathit{min}^t, \mathit{min}^s)$ and $\mathit{min}^{\{t\}} = \{\}$. It is clear how to define succ for pairs. To define succ for set types, consider a set $X : \{t\}$. Let a be the minimal element of type t which is not in X, and let $X_0 = \{x \in X \mid x \leq^t a\}$ and $X_1 = \{x \in X \mid a \leq^t x\}$. We claim $X_1 \cup a = \mathit{succ}^{\{t\}}(X)$. Clearly, $X \leq^{\{t\}} X_1 \cup a$. Let $X \leq^{\{t\}} Y$. Let $a \in Y$. Assume $x \in X_1 \cup a - Y = X_1 - Y$. Then $x \in X - Y$ and there exists $y \in Y - X$ such that $x \leq^t y$. Since $y \neq a$, we obtain $y \in Y - (X_1 \cup a)$ and $X_1 \cup a \leq^{\{t\}} Y$. If $a \notin Y$ and $x \in X_1 \cup a - Y$, then $Y - (X_1 \cup a) = Y - X_1$. Therefore, if $x \in X_1 - Y$, there exists $y \in Y - X_1$ such that $x \leq^t y$. Let $y_m = \max(Y - X_1)$. If $y_m \leq^t a$, then $Y - X_1 \subseteq X_0$ and $Y \subseteq X$ which contradicts our assumption. Then, since $a \notin Y$, $a \leq^t y_m$ and therefore $X_1 \cup a \leq^{\{t\}} Y$. This proves our claim. Therefore, $\mathit{succ}^{\{t\}}(X)$ is $\{\mathit{min}^t\} \cup X$ if $\mathit{min}^t \notin X$ and $X_1 \cup a$ if $\mathit{min}^t \in X$ where $a = \mathit{succ}^t(x_0)$ and $x_0 = \min\{x \in X \mid \mathit{succ}^t(x) \notin X\}$ and X_1 is defined as above. The expressibility of $\mathit{succ}^{\{t\}}$ follows immediately. □

4 The main result

We first define the notion of internal and generic family of functions. Then we show that the conservative extension property of $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =)$ endowed with well-founded linear orders can be preserved in the presence of any such family of functions. Introduce type variables α_i and consider nonground complex object types

$$\sigma, \tau ::= \alpha \mid b \mid \mathbb{B} \mid \mathbb{N} \mid \mathit{unit} \mid \sigma \times \tau \mid \{\sigma\}$$

If α_1, ..., α_n occur in σ, then $\sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n]$ stands for the type obtained by replacing every occurrence of α_i in σ by s_i. A complex object type s is an instance of a nonground complex object type σ if there are complex object types s_1, ..., s_n such that $s = \sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n]$ where α_1, ..., α_n are all the type variables in σ. The minimal height $mht(\sigma)$ of type σ is defined as the depth of nesting of set brackets in σ. That is, $mht(\sigma)$ is equivalent to $ht(s)$ where s is obtained from σ by replacing all occurrences of type variables in σ by unit. Let $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$ stand for a family of functions $p^{s_1,\ldots,s_n} : s \to t$ where $s = \sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n]$ and $t = \tau[s_1/\alpha_1, \ldots, s_n/\alpha_n]$. (Note that for each s_1, ..., s_n, there is exactly one $p^{s_1,\ldots,s_n}$ in the family $p^{\alpha_1,\ldots,\alpha_n}$.) The minimal height $mht(p)$ of $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$ is defined as $\max(mht(\sigma), mht(\tau))$.

Let $s = \sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n, t/\alpha]$. Let $dom^{s,t}_{\sigma,\alpha}(o)$ be the set of subobjects of type t in the object $o : s$ occurring at positions corresponding the the type variable α. Formally, define $dom^{s,t}_{\sigma,\alpha} : s \to \{t\}$ as follows: $dom^{s,t}_{b,\alpha}(x) = \{\}$; $dom^{s,t}_{\alpha,\alpha}(x) = \{x\}$; $dom^{s,t}_{\alpha',\alpha}(x) = \{\}$, where α and α' are distinct type variables; $dom^{u\times v,t}_{\sigma\times\tau,\alpha}(x, y) = dom^{u,t}_{\sigma,\alpha}(x) \cup dom^{v,t}_{\tau,\alpha}(y)$; and $dom^{\{s\},t}_{\{\sigma\},\alpha}(X) = \bigcup\{dom^{s,t}_{\sigma,\alpha}(x) \mid x \in X\}$.

Definition 4.1 The family of functions $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$ is *internal* (see [4]) in α_i if for all complex object types $s = \sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n]$, $t = \tau[s_1/\alpha_1, \ldots, s_n/\alpha_n]$, and complex object $o : s$, it is the case that $dom^{t,s_i}_{\tau,\alpha_i}(p^{s_1,\ldots,s_n}(o)) \subseteq dom^{s,s_i}_{\sigma,\alpha_i}(o)$. □

In other words, $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$ is internal in α_i if it does not invent new values in positions corresponding to the type variable α_i.

Let $s = \sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n, t/\alpha]$, $r = \sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n, t'/\alpha]$, and $\psi : t \to t'$. Let $modulate^{s,t,t'}_{\sigma,\alpha,\psi}(O)$ be the object $O' : r$ obtained by replacing every subobject $o : t$ in $O : s$ occurring in positions corresponding to type variable α by $\psi(o) : t'$. Formally, define $modulate^{s,t,t'}_{\sigma,\alpha,\psi} : s \to r$ as follows: $modulate^{s,t,t'}_{b,\alpha,\psi}(x) = x$; $modulate^{s,t,t'}_{\alpha,\alpha,\psi}(x) = \psi(x)$; $modulate^{s,t,t'}_{\alpha',\alpha,\psi}(x) = x$, where α and α' are distinct type variables; $modulate^{u\times v,t,t'}_{\sigma\times\tau,\alpha,\psi}(x, y) = (modulate^{u,t,t'}_{\sigma,\alpha,\psi}(x), (modulate^{v,t,t'}_{\tau,\alpha,\psi}(y))$; $modulate^{\{s\},t,t'}_{\{\sigma\},\alpha,\psi}(X) = \{modulate^{s,t,t'}_{\sigma,\alpha,\psi}(x) \mid x \in X\}$.

Definition 4.2 The family of functions $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$ is generic in α_i if for all complex object types $s = \sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n]$, $t = \tau[s_1/\alpha_1, \ldots, s_n/\alpha_n]$, complex object $o : s$, set $R : \{r\}$, and $\psi : s_i \to r$ such that ψ is a bijection from $dom^{s,s_i}_{\sigma,\alpha_i}(o)$ to R and $\psi^{-1} : r \to s_i$ is its inverse when restricted to

$dom^{s,s_i}_{\sigma,\alpha_i}(o)$, it is the case that

$$\begin{array}{ccc} s & \xrightarrow{p^{s_1,\ldots,s_n}} & t \\ {\scriptstyle modulate^{s,s_i,r}_{\sigma,\alpha_i,\psi}}\downarrow & & \uparrow{\scriptstyle modulate^{t',r,s_i}_{\tau,\alpha_i,\psi^{-1}}} \\ s' & \xrightarrow[p^{s'_1,\ldots,s'_n}]{} & t' \end{array}$$

the above diagram, where $s'_j = s_j$ for $j \neq i$ and $s'_i = r$, commutes. □

A family $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$ is called internal generic if it is internal and generic in all type variables. We remark here that our notion of genericity is slightly different from that in [4]. In particular, generic does not imply internal, in contrast to [4].

Now we demonstrate that adding a internal and generic family $p^{\alpha_1,\ldots,\alpha_n}$ to $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$ does not destroy its conservative extension property. We assume that $\leq^b$: $b \times b \to \mathbb{B}$ is a well-founded linear order for every base type b. Consider $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, \bigsqcup)$ obtained by adding the following construct to $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$:

$$\frac{e_1 : s \quad e_2 : \{t\}}{\bigsqcup\{e_1 \mid x^t \in e_2\} : s}$$

where $\bigsqcup\{e_1 \mid x^t \in e_2\}$ is the greatest element in the set $\{e_1 \mid x^t \in e_2\}$ (it is min^s when the set is empty). Note that $\bigsqcup\{e_1 \mid x \in e_2\}$ is already definable in $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$ if $e_1 : \{s\}$, and can be treated as a syntactic sugar. It is clear that both $dom^{s,t}_{\sigma,\alpha}$ and $modulate^{s,t,t'}_{\sigma,\alpha,\psi}$ are definable in $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, \bigsqcup)$ whenever ψ is.

Proposition 4.3 *Let $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$ be a family of functions that is internal generic. Then $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, \bigsqcup)$ endowed with the family of primitives $p^{\alpha_1,\ldots,\alpha_n}$ has precisely the expressive power of $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, \bigsqcup)$ endowed with just the primitive $p^{\mathbb{N},\ldots,\mathbb{N}}$.*

Proof. For each $s = \sigma[s_1/\alpha_1, \ldots, s_n/\alpha_n]$ and $o : \{s_i\}$, define

- $\psi(o) \triangleq \lambda x. \bigsqcup\{\mathit{if}\ x = \pi_1\ y\ \mathit{then}\ \pi_2\ y\ \mathit{else}\ 0 \mid y \in sort(o)\}$ and
- $\psi^{-1}(o) \triangleq \lambda x. \bigsqcup\{\mathit{if}\ x = \pi_2\ y\ \mathit{then}\ \pi_1\ y\ \mathit{else}\ min^s \mid y \in sort(o)\}$,

where $sort : \{s_i\} \to \{s_i \times \mathbb{N}\}$ is as defined in corollary 3.3. $\psi(o)$ and $\psi^{-1}(o)$ are functions of type $s_i \to \mathbb{N}$ and $\mathbb{N} \to s_i$ respectively. Clearly, $\psi(o)$ when restricted to o is a bijection whose inverse is $\psi^{-1}(o)$.

Let $u_i = \sigma[\mathbb{N}/\alpha_1, \ldots, \mathbb{N}/\alpha_{i-1}, s_i/\alpha_i, \ldots s_n/\alpha_n]$ and $v_i = \tau[\mathbb{N}/\alpha_1, \ldots, \mathbb{N}/\alpha_{i-1}, s_i/\alpha_i, \ldots s_n/\alpha_n]$. Note that $s = u_1$ and $t = v_1$. Define

- $\psi_i(o) \triangleq modulate^{u_i,s_i,\mathbb{N}}_{\sigma,\alpha_i,\psi(dom^{s,s_i}_{\sigma,\alpha_i}(o))}$ and

- $\psi_i^{-1}(o) \triangleq modulate^{v_{i+1},\mathbb{N},s_i}_{\tau,\alpha_i,\psi^{-1}(dom^{s,s_i}_{\sigma,\alpha_i}(o))}$.

Then the following diagram commutes by induction on n and by the assumption that the family $p^{\alpha_1,\dots,\alpha_n}$ is internal and generic.

$$\begin{tikzcd} o : u_1 \arrow[r, "\psi_1(o)"] \arrow[d, "p^{s_1,\dots,s_n}"'] & \bullet : u_2 \arrow[r, dotted, no head] \arrow[d, "p^{\mathbb{N},s_2,\dots,s_n}"'] & \bullet : u_n \arrow[r, "\psi_n(o)"] \arrow[d, "p^{\mathbb{N},\dots\mathbb{N},s_n}"] & \bullet : u_{n+1} \arrow[d, "p^{\mathbb{N},\dots,\mathbb{N}}"] \\ \bullet : v_1 & \bullet : v_2 \arrow[l, "\psi_1^{-1}(o)"] \arrow[r, dotted, no head] & \bullet : v_n & \bullet : v_{n+1} \arrow[l, "\psi_n^{-1}(o)"] \end{tikzcd}$$

Hence $p^{s_1,\dots,s_n} = \lambda x.\psi_1^{-1}(x) \circ \cdots \circ \psi_n^{-1}(x) \circ p^{\mathbb{N},\dots,\mathbb{N}} \circ \psi_n(x) \circ \cdots \circ \psi_1(x)$. The right hand side is clearly expressible in $\mathcal{NRC}(\mathbb{N}, +, \cdot, -, \div, \sum, =, \leq, \bigsqcup, p^{\mathbb{N},\dots,\mathbb{N}})$. $\square$

Next we sketch the proof of the conservativity of $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, \bigsqcup)$.

Proposition 4.4 $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, \bigsqcup)$ *has the conservative extension property with fixed constant* 0. *Moreover, when endowed with any additional primitive* p*, it retains the conservative extension property with fixed constant* $ht(p)$.

Proof sketch. The proof of theorem 2.1 is based on rewriting expressions of the language to normal forms in which the e_2 in subexpressions of the form $\bigcup\{e_1 \mid x \in e_2\}$ are variables. Such rewrite system was exhibited in Wong [10]. In Libkin and Wong [5] additional rules for $\sum$ were given which guarantee that e_2 in any subexpression $\sum\{e_1 \mid x \in e_2\}$ is a variable. The rewrite system was shown to be strongly normalizing and conservativity was derived by analyzing the normal forms.

Now we add the following rewrite rules for $\bigsqcup$, assuming that the use of the construct $\bigsqcup\{e_1 \mid x \in e_2\}$ is restricted to the situation when the type of e_1 is not a set type (when $e_1 : \{s\}$, it is treated as a shorthand.)

- $\bigsqcup\{e \mid x \in \{\}\} \rightsquigarrow min$
- $\bigsqcup\{e_1 \mid x \in \{e_2\}\} \rightsquigarrow e_1[e_2/x]$
- $\bigsqcup\{e \mid x \in e_1 \cup e_2\} \rightsquigarrow \textit{if } \bigsqcup\{e \mid x \in e_1\} \leq \bigsqcup\{e \mid x \in e_2\} \textit{ then } \bigsqcup\{e \mid x \in e_2\} \textit{ else } \bigsqcup\{e \mid x \in e_1\}$
- $\bigsqcup\{e_1 \mid x \in \bigcup\{e_2 \mid y \in e_3\}\} \rightsquigarrow \bigsqcup\{\bigsqcup\{e_1 \mid x \in e_2\} \mid y \in e_3\}$
- $\bigsqcup\{e \mid x \in \textit{if } e_1 \textit{ then } e_2 \textit{ else } e_3\} \rightsquigarrow \textit{if } e_1 \textit{ then } \bigsqcup\{e \mid x \in e_2\} \textit{ else } \bigsqcup\{e \mid x \in e_3\}$
- $\pi_i \bigsqcup\{e_1 \mid x \in e_2\} \rightsquigarrow \bigcup\{\textit{if } \sum\{\textit{if } e_1 \leq e_1[y/x] \textit{ then } 1 \textit{ else } 0 \mid y \in e_2\} = 1 \textit{ then } \pi_i\, e_1 \textit{ else } \{\} \mid x \in e_2\}$, when $e_1 : \{s\}$.
- $\pi_i \bigsqcup\{e_1 \mid x \in e_2\} \rightsquigarrow \bigsqcup\{\textit{if } \sum\{\textit{if } e_1 \leq e_1[y/x] \textit{ then } 1 \textit{ else } 0 \mid y \in e_2\} = 1 \textit{ then } \pi_i\, e_1 \textit{ else } \{\} \mid x \in e_2\}$, when e_1 is not of set type.

The extended collection of rewrite rules forms a weakly normalizing rewrite system and conservativity can be derived by induction on the induced normal forms along the lines of Wong [10]. $\square$

Putting together the two previous propositions, the desired theorem follows straightforwardly.

Theorem 4.5 $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, \bigsqcup)$ *endowed with a internal generic family* $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$ *has the conservative extension property with fixed constant* $mht(p)$. $\square$

5 Some corollaries

As remarked earlier, $\bigsqcup\{e_1 \mid x \in e_2\}$ is already definable in $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$ if $e_1 : \{s\}$. Therefore, if every type variable occurs in the scope of some set brackets in σ and τ, then the assumption of well-foundedness on $\leq^b$ used in proposition 4.3 is not required and the proposition holds for $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$. Thus, we have

Corollary 5.1 $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$ *endowed with a internal generic family* $p^{\alpha_1,\ldots,\alpha_n} : \sigma \to \tau$, *where each type variable is within the scope of some set brackets, has the conservative extension property at all input and output heights with fixed constant* $mht(p)$. $\square$

In particular, any polymorphic function definable in the algebra of Abiteboul and Beeri [1], which is equivalent to $\mathcal{NRC}(=, powerset)$, gives rise to a internal generic family of functions for all possible instantiations of type variables. (Observe that this is not true for $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$ because $succ^{\mathbb{N}}$ is definable and therefore can be lifted to any type by corollary 3.4. This also indicates that the notion of internal and generic family is more general than polymorphic functions.) Now we conclude immediately that the following languages possess the conservative extension property (see also [5]):

- $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, tc)$ with fixed constant 1, and
- $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq, powerset)$ with fixed constant 2.

where $tc^s : \{s \times s\} \to \{s \times s\}$ is the transitive closure of binary relations. Since the Abiteboul and Beeri algebra has the power of a fixpoint logic, a great deal of polymorphic functions can be added to $\mathcal{NRC}(\mathbb{N}, +, \cdot, \dot{-}, \sum, =, \leq)$ without destroying its conservative extension property (but may be increasing the fixed constant).

Acknowledgements. We thank Peter Buneman, Rick Hull and Dan Suciu for helpful discussions. We are also grateful to the anonymous referees for the useful comments. Leonid Libkin was supported in part by NSF Grant IRI-90-04137 and AT&T Doctoral Fellowship and Limsoon Wong was supported in part by ARO Grant DAAL03-89-C-0031-PRIME.

References

[1] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. In *Proc. International Workshop on Theory and Applications of Nested Relations and Complex Objects*, Darmstadt, 1988.

[2] G. Birkhoff. *Lattice Theory.* American Mathematical Society, 3rd edition, 1967.

[3] V. Breazu-Tannen, P. Buneman, and L. Wong. Naturally embedded query languages. In J. Biskup and R. Hull, editors, *LNCS 646: Proc. International Conference on Database Theory, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 1992.

[4] R. Hull. Relative information capacity of simple relational database schemata. *SIAM Journal of Computing*, 15(3):865–886, 1986.

[5] L. Libkin and L. Wong. Aggregate functions, conservative extension, and linear orders. In *Proceedings of 4th International Workshop on Database Programming Languages*, Springer Verlag, 1993, pages 279–292.

[6] A. Ohori, P. Buneman, and V. Breazu-Tannen. Database programming in Machiavelli: A polymorphic language with static type inference. In J. Clifford, et al., editors, *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 46–57, Portland, Oregon, June 1989.

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

[8] 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.

[9] W. Wechler. *Universal Algebra for Computer Scientists*, Springer-Verlag, Berlin, 1992.

[10] L. Wong. Normal forms and conservative properties for query languages over collection types. In *Proceedings of 12th ACM Symposium on Principles of Database Systems*, pages 26–36, Washington, D. C., May 1993.