Some Properties of Query Languages for Bags

Leonid Libkin* Limsoon Wong†

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

Abstract

In this paper we study the expressive power of query languages for nested bags. We define the ambient bag language by generalizing the constructs of the relational language of Breazu-Tannen, Buneman and Wong, which is known to have precisely the power of the nested relational algebra. Relative strength of additional polynomial constructs is studied, and the ambient language endowed with the strongest combination of those constructs is chosen as a candidate for the basic bag language, which is called $\mathcal{BQL}$ (Bag Query Language). We prove that achieveing the power of $\mathcal{BQL}$ in the relational language amounts to adding simple arithmetic to the latter. We show that $\mathcal{BQL}$ has shortcomings of the relational algebra: it can not express recursive queries. In particular, parity test is not definable in $\mathcal{BQL}$. We consider augmenting $\mathcal{BQL}$ with powerbag and structural recursion to overcome this deficiency. In contrast to the relational case, where powerset and structural recursion are equivalent, the latter is stronger than the former for bags. We discuss problems with using structural recursion and suggest a new bounded loop construct which works uniformly for bags, sets and lists. It has the power of structural recursion and does not require any preconditions to be verified. We find relational languages equivalent to $\mathcal{BQL}$ with powerbag and structural recursion/bounded loop. Finally, we discuss orderings on bags for rigorous treatment of partial information.

1 Summary

Sets and bags are closely related structures. While sets have been studied intensively by the theoretical database community, bags have not received the same amount of attention. However, real implementations frequently use bags as the underlying data model. For example, the "select distinct" construct and the "select average of column" construct of SQL can be better explained if bags instead of sets are used. In an earlier paper [5], Breazu-Tannen, Buneman, and Wong defined a language based on monads [20, 29] and structural recursion [3] for querying sets. In section 2 of this report, the same syntax is given a bag-theoretic semantics. We use this language as our ambient bag language

*Supported in part by NSF Grant IRI-90-04137 and AT&T Doctoral Fellowship.

†Supported in part by NSF Grant IRI-90-04137 and ARO Grant DAALO3-89-C-0031-PRIME.

and study its properties. Due to space limitations, we give only sketches of some of the proofs. Full proofs can be found in [18].

The ambient bag language is inadequate in expressive power as it stands; for example, it can not express duplicate elimination. In section 3, additional primitives are proposed and their relative strength with respect to the ambient language is fully investigated. The primitive *unique* which eliminates duplicates from a bag is shown to be independent of the other primitives. A similar result was obtained by Van den Bussche and Paredaens in the setting of pure object oriented databases [8]. The primitive *monus* which subtracts one bag from another is proved to be the strongest amongst the remaining primitives. This result was independently obtained by Albert [2]. However, his investigation on relative strength is not as complete as this report. As a consequence, we regard the ambient language augmented with *monus* and *unique* as our basic bag language. This language will be called $\mathcal{BQL}$ (Bag Query Language).

The relationship between bag and set queries is studied in Section 4. It is shown that the class of set functions computed by the ambient bag language endowed with equality on base types, an emptiness test, and *unique*, is precisely the class of functions computed by the nested relational language of [5]. Furthermore, if equality at all types is available, then the former strictly includes the latter. Grumbach and Milo also examined the relationship between sets and bags [9]. However they considered set functions on relations whose height of set nesting is at most 2. No such limit is imposed in this report.

The relationship between sets and bags can be examined from a different perspective. In the remainder of section 4, we investigate augmenting the set language of [5] to endow it with precisely the expressive power of our basic bag language $\mathcal{BQL}$. This is achieved by adding natural numbers, multiplication, subtraction, and a summation construct to the nested relational language. This also illustrates the natural relationship between bags and numbers.

In section 5, we use the connection to nested relational language established in section 4 to prove several fundamental properties of $\mathcal{BQL}$. In particular, the inexpressibility of properties (such as parity test) on natural numbers that are simultaneously infinite and co-infinite.

Breazu-Tannen, Buneman, and Wong proved that the power of structural recursion on sets can be obtained by adding a powerset operator to their language [5]. However, this result is contingent upon the restriction that every type has a finite domain. In section 6, the powerbag primitive of Grumbach and Milo [9] is contrasted with structural recursion on bags. In particular, the latter is shown to be strictly more expressive than the former. Although a powerbag primitive increases expressive power considerably, it is difficult to express algorithms that are efficient. While structural recursion does not have this deficiency, it requires the satisfaction of certain preconditions that cannot be automatically verified [4]. In section 6, a bounded loop construct which does not require the verification of any precondition is introduced. It is shown to be equivalent in expressive power to structural recursion over sets, bags, as well as lists. This confirms the intuition that structural recursion is just a special case of bounded loop. Furthermore, in contrast to the powerbag primitive which gives us all elementary functions [9], structural recursion gives us all primitive recursive functions. Also in section 6 we show that nonpolynomial operations on bags are more powerful than their set analogs, and find the primitive that precisely fills the gap.

Finally, in section 7, we show how to extend the approach of Buneman, Jung and Ohori [6] and Libkin [16] that uses certain partial orders to give semantics of databases with partial information to bags. We extend the idea of Libkin and Wong [18] of defining an ordering whose meaning is "being more partial". Such an ordering is fully characterized for bags, and we demonstrate an efficient algorithm to test it.

Related work. The semantic aspects of programming with collections using structural recursion were studied by Breazu-Tannen and Subrahmanyam in [4]. In particular, they showed that certain preconditions have to be satisfied for structural recursion to be well defined. Breazu-Tannen, Buneman and Naqvi brought out the connection between structural recursion and database query languages [3]. Breazu-Tannen, Buneman and Wong avoided the need of checking preconditions by placing a simple syntactic restriction on structural recursion [5]. The language so restricted has several equivalent formulations, one of them being $\mathcal{NRC}$ [5, 30]. This language is equivalent to the algebra of Abiteboul and Beeri [1] without the powerset operator.

Then Wong [30] proved that the language has the conservative extension property at all input/output heights. That is, the expressive power of the language is independent of the height of set nesting in the intermediate data. Then Libkin and Wong [19] showed that in the presence of very simple arithmetic operators conservativity can be extended uniformly to all input/output heights for languages augmented with bounded fixpoint operator, transitive closure, powerset and many other operators.

In [17] Libkin and Wong extended the use of the language $\mathcal{NRC}$ for querying or-sets. Grumbach and Milo [9] applied the algebra of Abiteboul and Beeri to bags. In particular, they investigated the relationship between set and bag languages restricted to certain input/output heights and the expressive power of bag languages with respect to the level of bag nesting. The basic bag language proposed in this report ($\mathcal{BQL}$) is precisely the language of Grumbach and Milo without the powerbag operator. Vickers [28] studied refinements of bags which are a more general concept than the ordering we introduce in this paper. In particular, our ordering can be expressed as a refinement, but there exist certain refinements of bags which lead to counterintuitive results when applied in the study of partial information.

The expressive power of Datalog under set and bag semantics was compared in [21]. In particular, an example of query was given that can not be expressed under the former but can be expressed under the latter. In [27] Saraiya shows that Datalog can be simulated with structural recursion on sets, preserving the PTIME complexity, by using as an intermediate step the loop operator described in section 6.2, and proving in the process that loop can be simulated by structural recursion (half of theorem 6.3 below). Several complexity-theoretic results for program properties and transformations are then be obtained by recourse to known results for Datalog.

2 The ambient nested bag language

The nested relational language proposed by Breazu-Tannen, Buneman, Wong [5] is denoted by $\mathcal{NRL}$ here. We now define an ambient bag query language $\mathcal{NBL}$. It is obtained by replacing the set constructs in $\mathcal{NRL}$ by the corresponding bag constructs. The language has two presentations – algebraic, called $\mathcal{NBA}$, and calculus style, called $\mathcal{NBC}$ – which are equivalent in terms of expressive power.

Types. The types in $\mathcal{NBL}$ are either complex object types or are function types $s \to t$ where s and t are complex object types. These types are the same as those of $\mathcal{NRL}$ except that bags $\{\!|s|\!\}$ instead of sets $\{s\}$ are used. The grammar for complex object types is given below.

$$s ::= b \mid unit \mid s \times s \mid \{\!|s|\!\}$$

A complex object type denotes a set of objects. *unit* is a special base type having exactly one element which we denote by $()$. $s \times t$ is the set of pairs whose first component is from s and whose second component is from t. $\{\!|s|\!\}$ are finite bags containing elements of type s. A bag is different from a set in that it is sensitive to the number of times an element occurs in it while a set is not. Finally, b are base types to be specified.

Expressions. The expressions of $\mathcal{NBA}$ and $\mathcal{NBC}$ are given in figure 1. The type superscripts are usually omitted as they can be inferred [13, 23]. The semantics of these constructs is similar to the semantics of $\mathcal{NRL}$ except duplicates are not eliminated. Semantics of $\mathcal{NBA}$ constructs is as follows. Kc is the constant function that produces the constant c. id is the identity function. $g \circ h$ is the composition of functions g and h; that is, $(g \circ h)(d) = g(h(d))$. The bang ! produces $()$ on all inputs. π_1 and π_2 are the two projections on pairs. $\langle g, h \rangle$ is pair formation; that is, $\langle g, h\rangle(d) = (g(d), h(d))$. $K\{\!||\!\}$ produces the empty bag. $\uplus$ is the additive bag union. b_η forms singleton bags: $b\_\eta(x) = \{\!|x|\!\}$. b_μ flattens a bag of bags: $b\_\mu\{\!|B_1, \ldots, B_n|\!\} = B_1 \uplus \ldots \uplus B_n$. $b\_map(f)$ applies f to every item in the input bag. Function b_ρ_2 is used for interaction between bags and pairs: $b\_\rho_2(x, y)$ pairs x with every item in the bag y. For example, $b\_\rho_2(1, \{\!|1, 2|\!\})$ returns $\{\!|(1, 1), (1, 2)|\!\}$.

Semantics of the $\mathcal{NBC}$ constructs which differ from $\mathcal{NBA}$ constructs is as follows. $\{\!||\!\}$ is the empty bag. $\{\!|e|\!\}$ is the singleton bag containing e. $\biguplus\{\!|e_1 \mid x \in e_2|\!\}$ is the bag obtained by first applying the function $\lambda x.e_1$ to each item in the bag e_2 and then taking the bag union of the results. For example, $\biguplus\{\!|\{\!|x, x+1|\!\} \mid x \in \{\!|1, 2, 3|\!\}|\!\}$ evaluates to $\{\!|1, 2, 2, 3, 3, 4|\!\}$.

Proposition 2.1 *The languages $\mathcal{NBA}$ and $\mathcal{NBC}$ have the same expressive power.* □

Therefore, we normally work with the component that is most convenient.

EXPRESSIONS OF $\mathcal{NBA}$

Category with Products

$$\frac{}{Kc : unit \to b} \qquad \frac{}{id^s : s \to s} \qquad \frac{h : r \to s \quad g : s \to t}{g \circ h : r \to t} \qquad \frac{}{!^s : s \to unit}$$

$$\frac{}{\pi_1^{s,t} : s \times t \to s} \qquad \frac{}{\pi_2^{s,t} : s \times t \to t} \qquad \frac{g : r \to s \quad h : r \to t}{\langle g, h \rangle : r \to s \times t}$$

Bag Monad

$$\frac{}{b\_\eta^s : s \to \{\!|s|\!\}} \qquad \frac{}{b\_\mu^s : \{\!|\{\!|s|\!\}|\!\} \to \{\!|s|\!\}}$$

$$\frac{f : s \to t}{b\_map(f) : \{\!|s|\!\} \to \{\!|t|\!\}} \qquad \frac{}{K\{\!||\!\}^s : unit \to \{\!|s|\!\}}$$

$$\frac{}{\uplus^s : \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|s|\!\}} \qquad \frac{}{b\_\rho_2^{s,t} : s \times \{\!|t|\!\} \to \{\!|s \times t|\!\}}$$

EXPRESSIONS OF $\mathcal{NBC}$

Lambda Calculus and Products

$$\frac{}{c : b} \qquad \frac{}{x^s : s} \qquad \frac{e : t}{\lambda x^s.e : s \to t} \qquad \frac{e_1 : s \to t \quad e_2 : s}{e_1\ e_2 : t}$$

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

Bag Monad

$$\frac{}{\{\!||\!\}^s : \{\!|s|\!\}} \qquad \frac{e : s}{\{\!|e|\!\} : \{\!|s|\!\}} \qquad \frac{e_1 : \{\!|s|\!\} \quad e_2 : \{\!|s|\!\}}{e_1 \uplus e_2 : \{\!|s|\!\}}$$

$$\frac{e_1 : \{\!|t|\!\} \quad e_2 : \{\!|s|\!\}}{\biguplus \{\!|e_1 \mid x^s \in e_2|\!\} : \{\!|t|\!\}}$$

Figure 1: Syntax of $\mathcal{NBL}$

3 Relative strength of bag operators

Breazu-Tannen, Buneman, and Wong [5] added equality test eq_s for all types s to $\mathcal{NRL}$. They showed that the presence of equality tests elevates $\mathcal{NRL}$ from a language that merely has structural manipulation capability to a full fledged nested relational language. The question of what primitives to add to $\mathcal{NBL}$ to make it a useful nested bag language should now be considered.

Unlike languages for sets for which we have a well established yardstick, very little is known about bags. Due to this lack of an adequate guideline, a large number of primitives are considered. Let us first fix some meta notations. A bag is just an *unordered* collection of items. $count(d, B)$ is defined to be the number of times the object d occurs as an element in the bag B. The bag operations to be considered are listed below.

- $monus : \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|s|\!\}$. $monus(B_1, B_2)$ evaluates to a B such that for every $d : s$, $count(d, B) = count(d, B_1) - count(d, B_2)$ if $count(d, B_1) > count(d, B_2)$; and $count(d, B) = 0$ otherwise.

- $max : \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|s|\!\}$. $max(B_1, B_2)$ evaluates to a B such that for every $d : s$, $count(d, B) = \max(count(d, B_1), count(d, B_2))$.

- $min : \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|s|\!\}$. $min(B_1, B_2)$ evaluates to a B such that for every $d : s$, $count(d, B) = \min(count(d, B_1), count(d, B_2))$.

- $eq : s \times s \to \{\!|unit|\!\}$. $eq(d_1, d_2) = \{\!|()|\!\}$ if $d_1 = d_2$; it evaluates to $\{\!||\!\}$ otherwise. That is, we are simulating booleans as a bag of type $\{\!|unit|\!\}$. True is represented by the singleton bag $\{\!|()|\!\}$ and False is represented by the empty bag $\{\!||\!\}$.

- $member : s \times \{\!|s|\!\} \to \{\!|unit|\!\}$. $member(d, B) = \{\!|()|\!\}$ if $count(d, B) > 0$; it evaluates to $\{\!||\!\}$ otherwise.

- $subbag : \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|unit|\!\}$. $subbag(B_1, B_2) = \{\!|()|\!\}$ if for every $d : s$, $count(d, B_1) \leq count(d, B_2)$; it evaluates to $\{\!||\!\}$ otherwise.

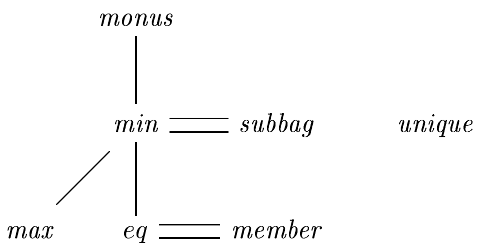
- $unique : \{\!|s|\!\} \to \{\!|s|\!\}$. $unique(B)$ eliminates duplicates from B. That is, for every $d : s$, $count(d, B) > 0$ if and only if $count(d, unique(B)) = 1$.

Each of these operators has polynomial time complexity with respect to size of input. Hence every function definable in $\mathcal{NBL}(monus, max, min, eq, member, subbag, unique)$, where we have explicitly listed the additional primitives in brackets, has polynomial time and space complexity with respect to the size of input.

The expressive power of these primitives relative to $\mathcal{NBL}$ is compared here. In contrast to $\mathcal{NRL}$, where all nonmonotonic primitives are interdefinable [5], these bag primitives differ considerably in expressive power. As a consequence of the theorem below, $\mathcal{NBL}(monus, unique)$ can be considered as the most powerful candidate for a standard bag query language. We denote $\mathcal{NBL}(monus, unique)$ by $\mathcal{BQL}$.

Theorem 3.1 *monus can express all primitives other than unique. unique is independent of the rest of the primitives. min is equivalent to subbag and can express both max and eq. member and eq are interdefinable and both are independent of max.* □

The results of theorem 3.1 can be visualized in the following diagram.



The independence of *unique* was also proved by Van den Bussche and Paredaens [8] and the fact that *monus* is the strongest amongst the remaining primitives was also showed by Albert [2]. However, their comparison was incomplete. For example, the incomparability of *max* and *eq* was not reported. In contrast, the results presented in this section can be put together in theorem 3.1 which completely and strictly summarizes the relative strength of these primitives.

4 Relationship between bags and sets

In this section, we study the relationship between bags and sets from two perspectives. First, we find a bag language whose set theoretic expressive power is that of $\mathcal{NRL}(eq)$. Then we consider endowing $\mathcal{NRL}(eq)$ with new primitives that would give it precisely the expressive power of the basic bag language $\mathcal{BQL}$.

4.1 Set-theoretic expressive power of bag languages

Several fragments of our nested bag language are compared with the nested relational language $\mathcal{NRL}(eq)$. This can be regarded as an attempt to understand the "set theoretic" expressive power of these bag languages. In order to compare bags and sets, two technical devices are required for conversions between bags and sets. We use the following constructs for this purpose:

$$\frac{f : s \to t}{bs\_map(f) : \{\!|s|\!\} \to \{t\}} \qquad \frac{f : s \to t}{sb\_map(f) : \{s\} \to \{\!|t|\!\}}$$

The semantics is as follows. $bs\_map(f)(R)$ applies f to every item in the bag R and then puts the results into a set. For example, $bs\_map(\lambda x.1{+}x)\{\!|1,2,3,1,4|\!\}$ returns the set $\{2,3,4,5\}$. $sb\_map(f)(R)$ applies f to every item in the set R and then puts the results into a bag. For example, $sb\_map(\lambda x.4)\{1,2,3\}$ returns the bag $\{\!|4,4,4|\!\}$.

Let s be a complex object type not involving bags. Then $\mathit{to\_bag}(s)$ is a complex object type obtained by converting all set brackets in s to bag brackets. Every object o of type s is converted to an object $\mathit{to\_bag}_s(o)$ of type $\mathit{to\_bag}(s)$. Conversely, let s be a complex object type not involving sets. Then $\mathit{from\_bag}(s)$ is a complex object type obtained by converting all bag brackets in s to set brackets. Every object o of type s is converted to an object $\mathit{from\_bag}_s(o)$ of type $\mathit{from\_bag}(s)$. The conversion operations are given inductively below.

$$
\begin{aligned}
&\mathit{to\_bag}_{\mathit{unit}} := \lambda x.x\\
&\mathit{to\_bag}_{s\times t} := \lambda x.(\mathit{to\_bag}_s(\pi_1\ x), \mathit{to\_bag}_t(\pi_2\ x))\\
&\mathit{to\_bag}_{\{s\}} := \mathit{sb\_map}(\mathit{to\_bag}_s)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{from\_bag}_{\mathit{unit}} := \lambda x.x\\
&\mathit{from\_bag}_{s\times t} := \lambda x.(\mathit{from\_bag}_s(\pi_1\ x), \mathit{from\_bag}_t(\pi_2\ x))\\
&\mathit{from\_bag}_{\{\!|s|\!\}} := \mathit{bs\_map}(\mathit{from\_bag}_s)
\end{aligned}
$$

Define $\mathcal{SET}(\Gamma)$ to be the class of functions $f : s \to t$ where s and t are complex object types not involving bags and Γ is a list of primitives such that there is $f' : \mathit{to\_bag}(s) \to \mathit{to\_bag}(t)$ definable in $\mathcal{NBL}(\Gamma)$ and the diagram below commutes.

$$
\begin{tikzcd}
\mathit{to\_bag}(s) \arrow[r, "f'"] & \mathit{to\_bag}(t) \arrow[r, "id"] & \mathit{to\_bag}(t) \arrow[d, "\mathit{from\_bag}_{\mathit{to\_bag}(t)}"] \\
s \arrow[u, "\mathit{to\_bag}_s"] \arrow[r, "f"'] & t \arrow[u, "\mathit{to\_bag}_t"'] \arrow[r, "id"'] & t
\end{tikzcd}
$$

Let eq_b be equality test restricted to base types. Let $\mathit{empty} : \{\!|\mathit{unit}|\!\} \to \{\!|\mathit{unit}|\!\}$ be a primitive such that it returns the bag $\{\!|()|\!\}$ when applied to the empty bag and returns the empty bag otherwise. Then

Theorem 4.1 *1. $\mathcal{SET}(\mathit{unique}, eq_b, \mathit{empty}) = \mathcal{NRL}(eq)$.*

2. $\mathcal{NRL}(eq) \subsetneq \mathcal{SET}(\mathit{unique}, eq)$

3. $\mathcal{NRL}(eq)$ and $\mathcal{SET}(\mathit{monus})$ are incomparable. □

The class $\mathcal{SET}(\Gamma)$ is precisely the class of "set theoretic" functions expressible in $\mathcal{NBL}(\Gamma)$. Consequently, the above results say that $\mathcal{NBL}(\mathit{unique}, eq_b, \mathit{empty})$ is *conservative* over $\mathcal{NRL}(eq)$ in the sense that it has precisely the same set theoretic expressive power. On the other hand, $\mathcal{NBL}(\mathit{unique}, eq)$ is a true extension over the set language. However, the presence of *unique* is in a technical sense essential for a bag language to be an extension of a set language.

4.2 A set language equivalent to $\mathcal{BQL}$

It was shown earlier that $\mathcal{BQL} = \mathcal{NBL}(\mathit{monus}, \mathit{unique})$ is the most powerful amongst the bag languages considered so far. From the foregoing discussion, this bag language is a true extension of $\mathcal{NRL}(eq)$. In this subsection, the relationship between sets and bags is studied from a different perspective. In particular, the *precise* amount of extra power $\mathcal{BQL}$ possesses over $\mathcal{NRL}(eq)$ is determined.

Let us endow $\mathcal{NRL}(eq)$ with natural numbers $\mathbb{N}$ together with multiplication, subtraction, and summation as defined below.

- $\cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. The semantics of $\cdot$ is multiplication of natural numbers.
- $\dot{-} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ (sometimes called *modified subtraction*). The semantics is as follows:

$$n \dot{-} m = \begin{cases} n - m & \text{if } n - m \geq 0 \\ 0 & \text{if } n - m < 0 \end{cases}$$

- $\sum g : \{s\} \to \mathbb{N}$ where $g : s \to \mathbb{N}$. The semantics is as follows: $\sum g \ \{o_1, \ldots, o_n\} = g(o_1) + \ldots + g(o_n)$.

In the sequel, the notaion $\mathcal{L} \simeq \mathcal{L}'$ means that two languages $\mathcal{L}$ and $\mathcal{L}'$ have the same expressive power. If $\mathcal{L}$ and $\mathcal{L}'$ have different type systems, this requires translations from one type system to another. In the following result, this is achieved by treating bags as sets of pairs element–number of occurrences.

Theorem 4.2 $\mathcal{BQL} \simeq \mathcal{NRL}(\mathbb{N}, \not\prec, \cdot, \dot{-}, \shortparallel)$. □

In summary, we have the following exact characterization of the relative strength between the basic bag language and the relational language of Breazu-Tannen, Buneman, and Wong: $\mathcal{NRL}(\mathbb{N}, \sum, \cdot, \dot{-}, \shortparallel) \simeq \mathcal{BQL}$ and $\mathcal{NRL}(eq) = \mathcal{SET}(unique,\ eq_b,\ empty)$. Klug [15] and Ozsoyoglu, Ozsoyoglu, and Matos [24] had to introduce aggregate functions by repeating them for every column position of a relation. That is, $aggregate_1$ is for column one, $aggregate_2$ is for column two, etc. Klausner and Goodman used a notion of hiding to explain the nature of aggregate functions in relational query languages [14]. In addition to projections, they introduced hiding operators that "hide" columns of a relation. Aggregate functions are then applied to the column that is left exposed. Hiding is different from projection. Let $R := \{(1, 2), (1, 3)\}$. Then projecting out column two on R gives $\{1\}$ while hiding column two on R gives $\{(1, [2]), (1, [3])\}$, where $[\cdot]$ signifies hidden values. The use of hiding to retain duplicates (since sets have no duplicate by definition) is a little clumsy. It is better to use bags. The $\sum$ primitive can be used to implement aggregate functions and should be seen as a generalization of their approaches.

5 Relationship between bags and numbers

As seen earlier, natural numbers are present in our nested bag language as objects of type $\{\!|unit|\!\}$, which we now write as $\mathbb{N}$. In this section, the relationship between bags and numbers is investigated in more detail. The equivalence between $\mathcal{BQL}$ and $\mathcal{NRL}(\mathbb{N}, \sum, \cdot, \dot{-}, \shortparallel)$ allows us to establish the following fundamental result.

Theorem 5.1 *Let $\mathcal{U}$ be a property of natural numbers. That is, $\mathcal{U} \subseteq \mathbb{N}$. Then membership in $\mathcal{U}$ can be expressed in $\mathcal{BQL}$ iff either $\mathcal{U}$ or $\mathbb{N} - \mathcal{U}$ is finite.*

Proof sketch: Assume there is an infinite and co-infinite property $\mathcal{U}$ of natural numbers that is expressible in $\mathcal{BQL}$. Then by theorem 4.2 a function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) = 1$ for $n \in \mathcal{U}$ and $f(n) = 0$ for $n \notin \mathcal{U}$ is expressible in $\mathcal{NRL}(\mathbb{N}, \not<, \cdot, \dot{-}, \shortparallel)$. In [19] we proved that expressions of $\mathcal{NRL}(\mathbb{N}, \not<, \cdot, \dot{-}, \shortparallel)$ are independent of the height of the intermediate data. Careful analysis of functions of type $\mathbb{N} \to \mathbb{N}$ that do not involve set constructs shows that they coincide with polynomials almost everywhere and hence can not have infinitely many roots, without being zero almost everywhere. $\square$

It is well known that the traditional relational languages cannot express parity test [7]. By the result of [30], it cannot be expressed in $\mathcal{NRL}(eq)$. It follows from the theorem we just proved that it remains inexpressible even in the greatly enhanced $\mathcal{NRL}(\mathbb{N}, \not<, \cdot, +, \dot{-}, \shortparallel)$ and hence not expressible in $\mathcal{BQL}$. From this many other inexpressibility results follow.

Corollary 5.2 *None of the following functions is expressible in $\mathcal{BQL}$:*

- *parity test;*
- *division by a constant;*
- *bounded summation;*
- *bounded product;*
- *$gen : \mathbb{N} \to \{\!|\mathbb{N}|\!\}$ given by $gen(n) = \{\!|0, 1, \ldots, n|\!\}$.* $\square$

Therefore, the arithmetic of our basic bag query language is very limited. In fact, its arithmetic power can be characterized. A unary function $f : \mathbb{N} \to \mathbb{N}$ is said to be *almost polynomial* if there exists a polynomial function $g : \mathbb{N} \to \mathbb{N}$ (that is, a function built from its argument and constants by using addition, subtraction and multiplication) and a number n such that $f(x) = g(x)$ for any $x \geq n$ (that is, f is g in all but finitely many points). The class of almost polynomial functions is denoted by $\mathcal{P}^{\approx}$.

Proposition 5.3 *$\mathcal{P}^{\approx}$ is the class of unary arithmetic functions expressible in $\mathcal{BQL}$.* $\square$

6 Power operators, bounded loop and structural recursion

Abiteboul and Beeri [1] suggested *powerset* as a new primitive for $\mathcal{NRL}(eq)$ to increase its expressive power. For instance, both parity test and transitive closure become expressible in $\mathcal{NRL}(eq, powerset)$. On the other hand, Breazu-Tannen, Buneman, and Naqvi [3] introduced structural recursion as an alternative means for increasing the horsepower of query languages.

It was shown in [5] that endowing $\mathcal{NRL}(eq)$ with a structural recursion primitive, which we denote by s_sri, or with the *powerset* operator yields languages that are equi-expressive. However, this is contingent upon the contrived restriction that the domain of each type is finite. Since every type has finite domain, this result has an important consequence. Suppose the domain of type $\{s\}$ has cardinality n. Then every use of *powerset* on an input of type $\{s\}$ can be safely replaced by a function that computes all subsets of a set having at most n elements. Such a function is easily definable in $\mathcal{NRL}(eq)$. Therefore, $\mathcal{NRL}(eq) \simeq \mathcal{NRL}(eq, s\_sri) \simeq \mathcal{NRL}(eq, powerset)$, if all types have finite domains. Hence the extra power of *s_sri* and *powerset* has effect only when there are types whose domains are infinite. Types such as natural numbers proved to be important in the earlier part of this report. Therefore, the relationship of structural recursion and power operators should be re-examined.

The syntax for the structural recursion construct on sets is

$$\frac{i : s \times t \to t \quad e : t}{s\_sri(i, e) : \{s\} \to t}$$

The semantics is $s\_sri(i, e)\{o_1, \ldots, o_n\} = i(o_1, i(o_2, i(\ldots, i(o_n, e)\ldots)))$, provided i satisfies certain preconditions [4]. In particular, it is commutative: $i(a, i(b, X)) = i(b, i(a, X))$ and idempotent: $i(a, i(a, X)) = i(a, X)$. *s_sri* is undefined otherwise. Breazu-Tannen, Buneman, and Naqvi [3] proved that efficient algorithms for computing functions such as transitive closure can be expressed using structural recursion. While structural recursion gives rise to efficient algorithms, its well-definedness precondition cannot be automatically checked by a compiler [4]. Therefore this approach is not completely satisfactory.

The *powerset* operator is always well defined. Unfortunately, algorithms expressed using *powerset* are often unintuitive and inefficient. For example, to find transitive closure of a binary relation $R : \{s \times s\}$, one finds the domain of R by taking union of first and second projections of R, takes powerset of cartesian product of the domain with itself and then selects all elements from this powerset which are transitive and contain R. Intersection of those elements is the transitive closure of R.

To the best of our knowledge, the problem of expressing a polynomial time transitive closure algorithm in $\mathcal{NRL}(eq, powerset)$ is still open. We do not advocate the elimination of every expensive operations from query languages. However, we believe that expressive power should not be achieved using expensive primitives. That is, if a function can be expressed using a polynomial time algorithm in some languages, then one should not be forced to define it using an exponential time algorithm. For this reason, *powerset* is not a good candidate for increasing expressive power.

This section has three main objectives. First, we endow $\mathcal{BQL}$ with the bag analogs of the powerset and structural recursion operators and we show that the former is strictly less expressive than the latter. Second, we suggest an efficient bounded loop primitive which captures the power of structural recursion but does not require any preconditions. Finally, we show that bag nonpolynomial operators are strictly more expressive than their set analogs, and we show that the analog of the *gen* primitive on sets fills the gap.

6.1 Powerset, powerbag and structural recursion

Grumbach and Milo [9], following Abiteboul and Beeri [1], introduced the *powerbag* operator into their nested bag language. The semantics of *powerbag* is the function that produces a bag of all subbags of the input bag. For example, $powerbag\{\!|1,1,2|\!\} = \{\!|\{\!||\!\}, \{\!|1|\!\}, \{\!|1|\!\}, \{\!|2|\!\}, \{\!|1,1|\!\}, \{\!|1,2|\!\}, \{\!|1,2|\!\}, \{\!|1,1,2|\!\}|\!\}$. They also defined the *powerset* operator on bags as $unique \circ powerbag$. For example, $powerset\{\!|1,1,2|\!\}$ is $\{\!|\{\!||\!\}, \{\!|1|\!\}, \{\!|2|\!\}, \{\!|1,1|\!\}, \{\!|1,2|\!\}, \{\!|1,1,2|\!\}|\!\}$. We do not consider *powerset* on bags further because of the following result.

Proposition 6.1 $\mathcal{BQL}(powerbag) \simeq \mathcal{BQL}(powerset)$.

Proof sketch. Suppose a bag B is given; then another bag B' can be constructed such that for any $a \in B$, B' contains a pair $(a, \{\!|a, \ldots, a|\!\})$ where the cardinality of the second component is $count(a, B)$. Let $B'' = unique(B')$; then B'' can be computed by $\mathcal{BQL}$. Now observe that changing the second component of every pair to its *powerset* and then $b\_map(b\_\rho_2)$ followed by flattening will give us a bag where each element $a \in B$ will be given a unique label. Now applying *powerset* to this bag followed by elimination of labels produces $powerbag(B)$. □

Structural recursion on bags is defined using the construct

$$\frac{e : t \quad i : s \times t \to t}{b\_sri(i, e) : \{\!|s|\!\} \to t}$$

It is required that i satisfy the commutativity precondition: $i(a, i(b, X)) = i(b, i(a, X))$, which can not be automatically verified [4]. Its semantics is similar to the semantics of s_sri. We want to show that it is strictly stronger than *powerbag*.

Theorem 6.2 $\mathcal{BQL}(powerbag) \subsetneq \mathcal{BQL}(b\_sri)$.

Proof sketch. First, *powerbag* can be expressed using b_sri, cf. [3]. Then it can be shown that any function in $\mathcal{BQL}(powerbag)$ produces outputs whose sizes are bounded by an elementary function on the size of the input, but in $\mathcal{BQL}(b\_sri)$ it is possible to define a function that on the input of size n produces the output of the hyperexponential size (where the height of the stack of powers depends on n) and hence can not be bounded by an elementary function. □

As an illustration of theorem 6.2, we characterize precisely the classes of arithmetic functions that both languages express. It also gives an alternative proof of theorem 6.2.

Theorem 6.3 *a) The class of functions* $f : \mathbb{N} \times \ldots \times \mathbb{N} \to \mathbb{N}$ *definable in* $\mathcal{BQL}(b\_sri)$ *coincides with the class of primitive recursive functions.*
b) The class of functions $f : \mathbb{N} \times \ldots \times \mathbb{N} \to \mathbb{N}$ *definable in* $\mathcal{BQL}(powerbag)$ *coincides with the class of Kalmar-elementary functions.* □

Similar results for other languages for bags or sets with built-in natural numbers were proved in [9, 12].

6.2 Bounded loop and structural recursion

As mentioned earlier, *powerbag* is not a good primitive for increasing the power of the language. It is not polynomial time and compels a programmer to use clumsy solutions for problems that can be easily solved in polynomial time. In addition, *powerbag* is weaker than structural recursion. On the other hand, *b_sri* is efficient [3] but its well definedness precondition can not be verified by a compiler [4]. In this section, we present a bounded loop construct

$$\frac{f : s \to s}{loop^t(f) : \{\!|t|\!\} \times s \to s}$$

Its semantics is as follows: $loop(f)(\{\!|o_1, \ldots, o_n|\!\}, o) = f(\ldots f(o)\ldots)$ where f is applied n times to o.

The bounded loop construct is more satisfactory as a primitive than *powerbag* and *b_sri* for several reasons. First, in contrast to *powerbag*, efficient algorithms for transitive closure, division, etc. can be described using it. For example, given $R : \{\!|s \times s|\!\}$, let $f_R : \{\!|s \times s|\!\} \to \{\!|s \times s|\!\}$ be the function whose semantics is $f_R(R') = R \circ R'$. Let $dom(R)$ be the domain of R. Then $loop(f_R)(dom(R), R)$ is the transitive closure of R. Second, it is very similar to the for-next-loop construct of familiar programming languages such as Pascal and Fortran. Third, in contrast to *b_sri*, it has no preconditions to be satisfied. Lastly, it has the same power as *b_sri*.

Theorem 6.4 (see also [27]) $\mathcal{BQL}(loop) \simeq \mathcal{BQL}(b\_sri)$.

Proof sketch. For one inclusion, observe that $loop(f)(n, e) = b\_sri(f \circ \pi_2, e)(n)$. For the reverse inclusion, given an input bag B, first generate all possible permutations of B (that is, all possible rank assignments to elements of B). It can be done in $\mathcal{BQL}(loop)$. Then, using *loop*, simulate *b_sri* for each rank assignment, assuming the ranks tell us the order in which elements are processed. Having done so, apply *unique* to the result. Hence, any function of type $s \to \{\!|t_1|\!\} \times \ldots \times \{\!|t_k|\!\}$ that is definable in $\mathcal{BQL}(b\_sri)$ is also definable in $\mathcal{BQL}(loop)$. If one of the types is not under the scope of the bag brackets, then in that position a singleton will be produced. □

Therefore replacing structural recursion by bounded loop eliminates the need for verifying any precondition. If the i in $b\_sri(i, e)$ is not commutative, the translation used in the proof simply produces a bag containing all possible outcomes of applying $b\_sri(i, e)$, depending on how elements of the input are enumerated. If i is commutative, then such a bag has one element which is *the* result of applying $b\_sri(i, e)$. Hence *b_sri* is really an optimized bounded loop obtained by exploiting the knowledge that i is commutative. Furthermore, *loop* coincides with structural recursion over sets, bags, and (with appropriately chosen primitives) lists. The implementation of $b\_sri(i, e)$ using the bounded loop construct given in the proof of theorem 6.4 has exponential complexity but the source of inefficiency is in computing all permutations in order to return all possible outcomes. If we are allowed to pick a particular order of application of i in $b\_sri(i, e)$, then more efficient implementations are possible (see the full paper [18]).

Theorem 6.4 also sheds some light on theorem 6.3 a). It is known that functions computable by a language that has an assignment statement and *for n do S* are precisely the primitive recursive functions [22]. It was also proved by Robinson and Gladstone that the primitive recursive functions are built from the initial functions by composition and iteration: $f(n, \vec{x}) = g^{(n)}(\vec{x})$, see [22]. Now we proved that the power of the structural recursion is precisely the power of the bounded loop, which is in essence the $for - do$ iteration or the iteration schema of Robinson and Gladstone. This is the intuitive reason why the class of functions definable by the structural recursion on bags coincides with the class of the primitive recursive functions.

6.3 Power operators and structural recursion on sets and bags

We have introduced power operators and structural recursion for sets and bags. In section 4.2 we also demonstrated how a set language can be extended to capture the power of our basic bag language: $\mathcal{BQL} \simeq \mathcal{NRL}(\mathbb{N}, \not\prec, \cdot, \dot{-}, \shortparallel)$. Under the translations of theorem 4.2, $n : \mathbb{N}$ is carried to a bag of n units: $\{\!|(), \ldots, ()|\!\}$. Consider the following primitive in the set language (cf. corollary 5.2):

$$gen : \mathbb{N} \to \{\mathbb{N}\}, \qquad \eth\ltimes(\ltimes) = \{\nvdash, \nVdash, \ldots, \ltimes\}$$

Under translations of theorem 4.2, it corresponds to the bag language primitive that takes a bag of n units and returns bag of bags containing i units for each $i = 0, 1, \ldots, n$. In other words, it is $powerset^{unit} = unique \circ powerbag^{unit}$. Observe that it remains a polynomial operation.

Having made this observation, we can formulate the first result of the section.

Theorem 6.5 *a)* $\mathcal{NRL}(\mathbb{N}, \not\prec, \cdot, \dot{-}, \shortparallel, powerset) \subsetneq \mathcal{BQL}(powerbag)$*;*
b) $\mathcal{NRL}(\mathbb{N}, \not\prec, \cdot, \dot{-}, \shortparallel, s\_sri) \subsetneq \mathcal{BQL}(b\_sri)$.

Proof sketch. Inclusion easily follows from theorem 4.2. To demonstrate strictness, observe that $powerset^{unit}$ is definable in both $\mathcal{BQL}(powerbag)$ and $\mathcal{BQL}(b\_sri)$. Hence, in view of theorem 6.2, it is enough to show that gen is not expressible in $\mathcal{NRL}(\mathbb{N}, \not\prec, \cdot, \dot{-}, \shortparallel, s\_sri)$. Define the size of an object as follows: size of an object of a base type is 1 and size of a pair or a set is sum of the sizes of the components. Then, it is possible to show that for any function f definable in $\mathcal{NRL}(\mathbb{N}, \not\prec, \cdot, \dot{-}, \shortparallel, s\_sri)$ there exists a primitive recursive function φ_f such that, if $f(i) = o$ and sizes of i and o are s_i and s_o, then $s_o \leq \varphi_f(s_i)$. Now assume that gen is definable. Let $n = \varphi_{gen}(1)$. Then $n + 1 = size(gen(n + 1)) \leq \varphi_{gen}(size(n + 1)) = n$. This contradiction shows that gen is not definable. $\square$

Now we have a problem of filling the gap between set and bag languages with power operators or structural recursion. It turns out that the gen primitive is sufficiently powerful to do the job. The following result is proved by extending translations of theorem 4.2.

Theorem 6.6 *a)* $\mathcal{NRL}(\mathbb{N}, \not<, \cdot, \dot{-}, \shortparallel, powerset, \eth\ltimes) \simeq \mathcal{BQL}(powerbag)$*;*
b) $\mathcal{NRL}(\mathbb{N}, \not<, \cdot, \dot{-}, \shortparallel, s\_sri, \eth\ltimes) \simeq \mathcal{BQL}(b\_sri)$. □

As another illustration of the power of the *gen* primitive, we show that it allows us to simplify the *loop* construct without considerably losing expressiveness of the language. We simplify the *loop* construct by defining $iter(f) : \{\!|unit|\!\} \to \{\!|unit|\!\}$ where $f : \{\!|unit|\!\} \to \{\!|unit|\!\}$ as $iter(f)(n) = f(f(\ldots(f(\{\!|\,|\!\}))\ldots))$ where f is applied n times.

Corollary 6.7 $\mathcal{BQL}(iter, powerset^{unit})$ *expresses all unary primitive recursive functions.* □

7 Orderings on bags

In the previous sections we have concentrated on comparing expressive power of set and bag languages. In this section we study another important problem where sets and bags differ considerably, that is, semantics of partial information.

We follow the idea of Buneman, Jung and Ohori [6] and Libkin [16], where databases were considered as subsets of certain partially ordered sets in order to provide rigorous mathematical treatment of partial information. The intuitive meaning of the ordering is "being more partial". In [6, 16] only sets were considered. A rather intuitive approach to defining the orderings was adopted in [6, 16], and later in Libkin and Wong [17] that approach was justified. However, it is not immediately clear how to generalize any of the orderings of [6, 16, 17] to bags, and hence additional study is needed. In this section we use techniques of [17] to define an ordering for *bags*. Even though the ordering appears somewhat awkward, we demonstrate an effective algorithm to test whether two bags are comparable.

As in [11, 6, 16], we assume that partiality can be expressed by means of a partial order on database objects. That is, $a \leq b$ expresses the fact that a is more partial than b or b is more informative than a. It was mentioned in [6] that many models of partial information can be captured by this very general scheme. This approach is also suitable for databases *without* partial information. In such a case, values of base types are totally unordered.

It is usually assumed that orders on the base types are given. For example, if base type is $\mathbb{N}_\perp$ whose values are natural numbers or null ($\perp$), the usual ordering is $\perp \leq n$ for any $n \in \mathbb{N}$ and any two distinct natural numbers are not comparable, see Gunter [10]. The ordering is then extended to pairs in the usual way. That is, $(x, y) \leq (x', y')$ iff $x \leq_1 x'$ and $y \leq_2 y'$. However, if one wants to extend the ordering to subsets of an ordered set, many possibilities arise. In [17] we tried to define an ordering by saying that a set X is less informative than a set Y if there is a sequence of simple updates, each leading to a more informative set. Dealing with sets, we defined the primitive updates as follows: $X \rightarrowtail (X - \{a\}) \cup X'$ where $a \le b$ for any $b \in X'$. Notice that if $a \notin X$, this is equivalent to augmenting X by X'.

To extend this idea to bags, recall that having a bag rather than a set means that each element of a bag represents an object and if there are many occurrences of some element, then at the moment certain objects are indistinguishable. This justifies the following definition. We say that a bag B_2 is more informative than a bag B_1 if B_2 can be obtained from B_1 by a sequence of updates of the following form: (1) an element a is removed from B_1 and is replaced by an element b such that b is more informative than a, or (2) an element b is added to the bag B_1. Formally, let $\langle D, \le \rangle$ be a partially ordered set. Let $\mathcal{P}^b_{\text{fin}}(D)$ be the set of all finite bags whose elements are in D. Then, for $B_1, B_2 \in \mathcal{P}^b_{\text{fin}}(D)$, $B_1 \rightsquigarrow B_2$ iff $B_2 = (B_1 \, monus \{\!|a|\!\}) \uplus \{\!|b|\!\}$ where $a \le b$ or $B_2 = B_1 \uplus \{\!|b|\!\}$. The transitive-reflexive closure of $\rightsquigarrow$ is denoted by $\unlhd$. That is, we say that B_1 is *less informative* than B_2 if $B_1 \unlhd B_2$.

As proved in [17], the ordering on *sets* obtained as the transitive-reflexive closure of $\rightarrowtail$ coincides with the *lower powerdomain ordering* [10] defined as

$$X \le^\flat Y \text{ iff } \forall x \in X.\ \exists y \in Y.\ x \le y$$

A similar construction can be used to characterize $\unlhd$. Let $\mathbb{N}^{\shortparallel}$ denote the totally unordered poset whose elements are natural numbers (the superscript is used to distinguish it from $\mathbb{N}$ which in this paper denotes natural numbers with the usual ordering). For a finite bag B and an injective map $\phi : B \to \mathbb{N}^{\shortparallel}$, which is sometimes called *labeling*, by $\phi(B)$ we denote the set $\{(b, \phi(b)) \mid b \in B\}$. In other words, ϕ assigns a unique label to each element of a bag. If $B \in \mathcal{P}^b_{\text{fin}}(D)$, the ordering on pairs (b, n) where $b \in B$ and $n \in \mathbb{N}^{\shortparallel}$ is the usual pair ordering; that is, $(b, n) \le (b', n')$ iff $b \le b'$ and $n = n'$.

Proposition 7.1 *The binary relation $\unlhd$ on bags is a partial order. Given two bags B_1, B_2, $B_1 \unlhd B_2$ iff there exist labelings ϕ and ψ on B_1 and B_2 respectively such that $\phi(B_1) \le^\flat \psi(B_2)$.* □

The lower powerdomain ordering $\le^\flat$ of sets can be effectively verified. Indeed, if two sets are given, there is an $O(n^2)$ time complexity algorithm to check if they are comparable. The description of $\unlhd$ given above seems to be somewhat awkward algorithmically. However, it is not much harder to test for.

Proposition 7.2 *There exists an $O(n^{5/2})$ time complexity algorithm that, given two bags B_1 and B_2 in $\mathcal{P}^b_{\text{fin}}(D)$, returns* true *if $B_1 \unlhd B_2$ and* false *otherwise.*

Proof sketch. The problem is reduced to finding a maximal matching in a certain bipartite graph whose size in linear in the sum of the sizes of the two given bags. Hence, it can be solved by the Hopcroft-Karp algorithm in $O(n^{5/2})$. □

There is a big difference between orders on sets and bags. While $X \le^\flat Y$ does not say anything about cardinality of X and Y, $B_1 \unlhd B_2$ implies that the cardinality of B_1 is less than or equal to the cardinality of B_2. This reflects our point of view that having a bag rather than a set stored in a database means that each element of a bag represents an object and having two or more occurrences of the same elements means that at the moment some objects are indistinguishable. Therefore, the cardinality can not be reduced in the process of obtaining more information.

8 Conclusion and further work

Many results on bags are presented in this report. A large combination of primitives have been investigated and the relative strength is determined. The relationship between bags and sets has been studied from two different perspectives. First, various bag languages are compared with a standard nested relational language to understand their set-theoretic expressive power. Second, the extra expressive power of bags is characterized accurately. The relationship between bags and natural numbers is studied. In particular, we show that properties that are simultaneously infinite and co-infinite are inexpressible. Finally, the relationship between structural recursion and the powerbag operator has been re-examined. The former is shown to be stronger than the latter. Then we introduce the bounded loop construct that captures the power of structural recursion but has the advantage of not requiring verification of any precondition. Moreover, we prove that structural recursion gives us all primitive recursive functions.

There are several conjectures we have not yet proved. Does adding *gen* give us precisely lower elementary functions [26]? Are functions such as testing whether a graph is a tree or testing connectivity or transitive closure expressible in the set language equivalent to $\mathcal{BQL}$? What is the expressive power of this set language augmented by transitive closure? We know, for example, that test for balanced binary trees can be expressed in this language, but can it express bounded fixpoint? When augmented with *gen*, how powerful is it?

Breazu-Tannen, Buneman and Wong [5], Libkin and Wong [17], and this paper studied the use of monads and structural recursion for querying sets, or-sets and bags respectively. We hope to extend this methodology to other collection types such as lists, arrays, etc.

Acknowledgements. Peter Buneman gave us the initial inspiration and provided many helpful suggestions. We also thank Val Breazu-Tannen, Jean Gallier, Dan Suciu, Bennet Vance, Steve Vickers and Scott Weinstein for valuable comments and suggestions.

References

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

[2] J. Albert. Algebraic properties of bag data types. In *VLDB 91*, pages 211–219.

[3] V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In *DBPL 91*, pages 9–19.

[4] V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with sets/bags/lists. In *LNCS 510: ICALP 91*, pages 60–75.

[5] V. Breazu-Tannen, P. Buneman, and L. Wong. Naturally embedded query languages. In *ICDT 92*, pages 140–154.

[6] P. Buneman, A. Ohori, and A. Jung. Using powerdomains to generalize relational databases. *Theoretical Computer Science*, 91:23–55, 1991.

[7] A. Chandra and D. Harel. Structure and complexity of relational queries. *JCSS*, 25:99–128, 1982.

[8] J. Van den Bussche and J. Paredaens. The expressive power of structured values in pure OODB. Technical Report 90-23, University of Antwerp, 1990. Extended abstract in *PODS 91.*

[9] S. Grumbach and T. Milo. Towards tractable algebras for bags. In *PODS 93*, pages 49–60.

[10] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* The MIT Press, 1992.

[11] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31:761–791, 1984.

[12] N. Immerman, S. Patnaik and D. Stemple, The expressiveness of a family of finite set languages, in *Proceedings of the 10th Symposium on Principles of Database Systems*, 1991, pages 37–52.

[13] L. A. Jategaonkar and J. C. Mitchell. ML with extended pattern matching and subtypes. In *Proceedings of ACM Conference on LISP and Functional Programming*, pages 198–211, Snowbird, Utah, July 1988.

[14] A. Klausner and N. Goodman. Multirelations: semantics and languages. In *VLDB 85*, pages 251–258.

[15] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, 29(3):699–717, 1982.

[16] L. Libkin. A relational algebra for complex objects based on partial information. In J. Demetrovics and B. Thalheim editors, *LNCS 495: Proceedings of Symposium on Mathematical Fundamentals of Database Systems, Rostock, May 1991*, pages 36–41. Springer-Verlag, 1991.

[17] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. In *PODS 93*, Washington, D. C., May 1993, pages 37–48. Full paper available as UPenn Technical Report MS-CIS-92-88.

[18] L. Libkin and L. Wong. Query languages for bags, Technical Report MS-CIS-93-36, University of Pennsylvania, 1993.

[19] L. Libkin and L. Wong. Aggregate functions, conservative extension, and linear orders. This volume.

[20] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[21] I. S. Mumick and O. Shmueli, How expressive if stratified aggregation, submitted.

[22] P. Odifreddi. *Classical Recursion Theory.* North Holland, 1989.

[23] A. Ohori, P. Buneman, and V. Breazu-Tannen. Database programming in Machiavelli: a polymorphic language with static type inference. In *SIGMOD 89*, pages 46–57.

[24] G. Ozsoyoglu, Z. M. Ozsoyoglu, and V. Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM TODS*, 12(4):566–592, 1987.

[25] 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.

[26] H. E. Rose. *Subrecursion: Functions and Hierarchies.* Clarendon Press, Oxford, 1984.

[27] Y. Saraiya, Fixpoints and optimizations in a language based on structural recursion on sets, Manuscript, December 1992.

[28] S. Vickers. Geometric theories and databases. In P. Johnstone and A. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *London Mathematical Society Lecture Notes*, pages 288–314. Cambridge University Press, 1992.

[29] P. Wadler. Comprehending monads. In *Proceedings of ACM Conference on Lisp and Functional Programming*, Nice, June 1990.

[30] L. Wong. Normal forms and conservative properties for query languages over collection types. In *PODS 93*, pages 26–36, Washington, D. C., May 1993. Full paper available as UPenn Technical Report MS-CIS-92-59.