Query Languages for Bags and Aggregate Functions*

Leonid Libkin
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA
Email: libkin@research.att.com

Limsoon Wong
Institute of Systems Science
Heng Mui Keng Terrace
Singapore 0511
Email: limsoon@iss.nus.sg

Abstract

Theoretical foundations for querying databases based on bags are studied in this paper. We fully determine the strength of many polynomial-time bag operators relative to an ambient query language. Then we obtain $\mathcal{BQL}$, a query language for bags, by picking the strongest combination of these operators. The relationship between the nested relational algebra and various fragments of $\mathcal{BQL}$ is investigated. The precise amount of extra power that $\mathcal{BQL}$ possesses over the nested relational algebra is determined. It is shown that the additional expressiveness of $\mathcal{BQL}$ amounts to adding aggregate functions to a relational language.

The expressive power of $\mathcal{BQL}$ and related languages is investigated in depth. We prove that these languages possess the conservative extension property. That is, the expressibility of queries in these languages is independent of the nesting height of intermediate data. Using this result, we show that recursive queries, such as transitive closure, are not definable in $\mathcal{BQL}$. A new tool for analyzing expressibility, called the bounded degree property, is also introduced and we show how it can be used on relational languages.

To enhance the expressiveness of $\mathcal{BQL}$, we consider non-polynomial primitives such as *powerbag*, structural recursion, and bounded loop. Structural recursion on bags is shown to be equivalent to the bounded loop operator and strictly more powerful than the *powerbag* primitive. We show that the class of numerical functions expressible in $\mathcal{BQL}$ augmented by structural recursion is precisely the class of primitive recursive functions.

1 Introduction

Most research on database query languages concentrated on languages for sets. Furthermore, many languages such as the relational algebra can only manipulate data and cannot produce new values. However, real implementations frequently use bags as the underlying data model. In addition, they provide aggregate functions that produce values which are not stored in a database. For example, SQL has the "select distinct" construct, which makes sense only for bags, and the "select average" construct, which generates new values.

*Invited paper to appear in *Journal of Computer and System Sciences*. Preliminary versions appeared in the *Proceedings of 13th ACM Symposium on Principles of Database Systems*, Minneapolis, May 1995, and in the *Proceedings of 4th International Workshop on Database Programming Languages*, New York, August 1993.

The main goal of this paper is to understand the difference between theoretical languages, like the relational algebra, and languages which are much closer to practical languages like SQL. We find out how the use of bags and the use of aggregate functions are related. We also find out which classical results on the expressiveness of set languages continue to hold for bags and which do not.

One of the claims of the paper is that considering bags instead of sets allows us to obtain a rational reconstruction of SQL. In some books, even a statement that SQL is equivalent to the relational algebra can be found. For example, Paredaens et al. [45] state that the tuple relational calculus and SQL have the same expressive power. While no one really knows what SQL is, since there are many different versions, it is widely accepted that *any* version of SQL has at least two features which are not present in the relational algebra:

- SQL provides a number of aggregate operators. According to Ullman [57], the five usual ones are AVG, COUNT, SUM, MIN, and MAX. Many versions of SQL provide others, such as the standard deviation STDDEV or variance VARIANCE.
- SQL allows a limited form of nesting by using the GROUP-BY construct. In particular, by combining these two features, one can write a query that computes the average salary in each department, as shown below.

```
SELECT     Dept, AVG(Salary)
FROM       Emp
GROUP-BY   Dept
```

Note that the semantics of aggregate functions assumes that the underlying structure is a bag rather than a set. In other words, elimination of duplicates may lead to wrong results if aggregates are present in a language. For example, to compute

$$\mathsf{AVG}(\Pi_{\text{Salary}}(\text{Employees}))$$

one cannot remove duplicates from $\Pi_{\text{Salary}}(\text{Employees})$. Suppose at least two employees have the same salary. Then the result of applying AVG to the *set* $\Pi_{\text{Salary}}(\text{Employees})$ will be wrong. The correct result is achieved by applying AVG to the *bag* $\Pi_{\text{Salary}}(\text{Employees})$ in which no duplicates are removed after projection on the Salary field is performed.

This example reiterates that one needs *bag semantics* for the correct evaluation of aggregate functions. Even though this has been common knowledge since SQL was first conceived, little effort has been devoted to understand the connection between aggregate operators and bag query languages. Adding aggregate operations to the relational algebra was considered by Klug [30] and then extended to complex objects in Ozsoyoglu et al. [43]. The approach adopted in these papers is the following. An aggregate function is introduced separately for each column. That is, there are functions AVG_1, AVG_2, etc. To compute the average of the B column, one would write $\mathsf{AVG}_{\mathsf{B}}(\mathsf{R})$. While this approach incorporates aggregate functions into standard languages such as the nested relational algebra, it is not completely satisfactory as it introduces too many new functions.

Another way to explain the semantics of SQL aggregate operators was given by Klausner and Goodman [29] using the concept of *hiding.* Whenever a projection is followed by an application of an aggregate function, the projection operation is interpreted as an operation to hide the missing columns. For example, projecting on the second column of $\{(a,6),(b,6),(c,12)\}$ would yield $\{([a],6),([b],6),([c],12)\}$, where $[\cdot]$ signifies those "hidden" values. Thus, in contrast to the relational projection, hiding keeps both occurrences of 6 by tagging them with the hidden values. Then applying AVG gives the correct result. Using hiding to retain duplicates is rather clumsy. It is much better to use bags in the first place.

More recently, there has been some activity in trying to identify a "standard" query language for bags, in the same spirit that the relational algebra is considered the standard language for sets. First, Albert [4] proposed a number of operations on bags and established some of their properties. Two years later, Grumbach and Milo [21] extended the algebra of Abiteboul and Beeri [1] to bags. At almost the same time, we proposed [35] a bag language, $\mathcal{BQL}$, that turned out to be equivalent to the polynomial-time fragment of the language of Grumbach and Milo. Perhaps the most important property of $\mathcal{BQL}$ is its close connection with relational languages and aggregate functions. In fact, it has precisely the power of the nested relational algebra augmented with a general template for producing aggregate functions.

$\mathcal{BQL}$ can be seen as a rational reconstruction of SQL. The simplicity of its operations and the full-compositionality of its syntax allows it to be analyzed. We use this language to answer several questions on the expressibility of bag languages and set languages with aggregate functions. In particular, we investigate the relative expressive power of various query languages for bags and we find out the exact extra power that comes with aggregate functions and bags.

It is well known that first-order logic-based languages, such as the relational algebra or the tuple relational calculus, cannot express recursive queries like transitive closure [28]. Intuitively, adding nesting and aggregates to these languages will not give them sufficient expressiveness to do something as radical as transitive closure. In fact, it is often claimed that SQL cannot express recursive queries. However, this claim has never been proved. The previous result that comes closest to proving this claim is Consens and Mendelzon [15], where an assumption that DLOGSPACE $\neq$ NLOGSPACE is made.

The flat relational algebra is an algebraization of first-order logic. Therefore, to answer questions on the expressive power of flat relational languages based on the relational algebra, one can use a rich body of results on first-order expressibility, as in Chandra and Harel [12], Fagin [17], and Gaifman [19]. However, calculi for nested relations are essentially higher-order logics, where very little is known about expressibility over finite structures. Therefore, new techniques are needed for analyzing languages for nested collections. A difficulty is that, even in simple queries, one can increase the level of nesting in intermediate data and then get a desired result by flattening or unnesting. Unless there is some restriction on doing this, there is very little hope for finding nice tools for analyzing the expressiveness of such languages.

Fortunately, queries of the nested relational algebra were shown to be independent of the height of set nesting in intermediate data. The first result of this kind was proved by Paredaens and Van Gucht [46] for queries over flat relations. It was later generalized by Wong [59] to arbitrary queries. Recently, we showed that it continues to hold in the presence of aggregate functions [36] and that it holds even in the presence of a large variety of polymorphic functions [37]. This property provides the simplifying tool we need to analyze our languages.

Since bag languages essentially add aggregate functions to relational languages, and hence have built-in arithmetic, it is hard to find a logic that captures them. Thus it is not clear what techniques can be used for proving results about expressive power. There are several conjectures on $\mathcal{BQL}$ and on the nested relational algebra, formulated by Grumbach and Milo [21] and Paredaens [44]:

Conjecture 1 (Grumbach and Milo) *The parity test is not definable in $\mathcal{BQL}$.*

Conjecture 2 (Grumbach and Milo) *The transitive closure is not definable in $\mathcal{BQL}$.*

Conjecture 3 (Paredaens) *The test for balanced binary trees is neither definable in the nested relational algebra nor in $\mathcal{BQL}$.*

A variant of Conjecture 1 concerns parity of natural numbers, rather than parity of the cardinality of a bag. In this paper, among other things, we prove all three conjectures.

Let us make a few observations before we outline the main results. In most cases when people conjecture that something like transitive closure is not expressible in a language, they actually mean that the language is incapable of expressing recursive queries. Transitive closure just happens to be the most famous example of a recursive query. However, it is not the simplest one. As shown by Immerman [24], the first-order logic extended with transitive closure captures the complexity class NLOGSPACE over ordered structures. There are possibly simpler classes and complete problems for them. For example, DLOGSPACE is captured by the first-order logic with *deterministic* transitive closure [24]. Therefore, if we could show that deterministic transitive closure is not expressible in a language that has at least the power of first-order logic, then many other inexpressibility results will be obtained for free; for instance, connectivity and transitive closure. We exhibit two queries which are at most as hard as deterministic transitive closure, one of them being the test for balanced binary trees, and show that they are not expressible in $\mathcal{BQL}$.

There is also a lack of uniformity in proving inexpressibility results. There are well-known tools, such as Ehrenfaucht-Fraïssé games, for proving first-order inexpressibility. However, applying them to any query whose inexpressibility is to be proved is a separate combinatorial problem, which is sometimes non-trivial. For the Paredaens conjecture, it is easier to use another technique called Hanf's lemma, as presented in Fagin et al. [18], but it still requires some combinatorial proof which no longer works if we ask for balanced ternary trees, 4-ary trees, etc. Note that 0/1 laws do provide a uniform technique for proving inexpressibility results. However, they are rather restrictive. For example, all properties of graphs considered in this paper can easily be shown to have asymptotic probability 0, and hence 0/1 laws are of no help.

In this paper, we demonstrate a uniform technique for proving various inexpressibility results for the nested relational calculus. This technique is a variant of Gaifman's locality theorem [19], but it is often easier to apply, because it is a statement about semantic properties of queries, rather than their syntactic representation.

Organization and summary of main results. In Section 2 we outline a recent approach [6, 8, 53, 9] to designing query languages for collection types. This approach is based on turning universal properties of collections into programming syntax. Applying it to complex objects, we obtain a *nested relational language* called $\mathcal{NRL}$. The language resulting from adding equality test to $\mathcal{NRL}$, denoted by $\mathcal{NRL}(eq)$, has the same expressive power as the nested relational algebra, originally developed in Thomas and Fischer [54], Colby [13], and Schek and Scholl [49]. $\mathcal{NRL}(eq)$ is a better language for the purpose of this paper than these older languages because it has simpler semantics and it is easily extensible with operations such as aggregate functions. A very important property of $\mathcal{NRL}(eq)$ is its conservativity [46, 59] — the class of $\mathcal{NRL}(eq)$-queries from flat relations to flat relations is precisely the class of relational algebra, or first-order definable queries.

Another advantage of $\mathcal{NRL}$ is its extensibility. In particular, it is easy to see what constructs should be used if bags are used instead of sets. Turning the set constructs into the bag constructs, we obtain the *nested bag language* called $\mathcal{NBL}$. In Section 3, we augment it with a number of polynomial-time computable primitives suggested previously in Albert [4], Grumbach and Milo [21], and Van den Bussche and Paredaens [56]. We fully characterize their relative expressive power. The strongest combination of these primitives includes the bag difference *monus* and duplicate elimination *unique*. We define the basic bag language $\mathcal{BQL}$ (*Bag Query Language*) as $\mathcal{NBL}$ enhanced with these two primitives.

In Section 4, the relationship of bag and set queries is studied. First, we prove the following theorem.

Theorem 4.1 *The class of set functions computed by $\mathcal{NBL}$ endowed with equality on base types, test for emptiness, and duplicate elimination is precisely the class of functions computed by $\mathcal{NRL}$.*

The relationship between sets and bags is also examined from a different perspective. We add enough machinery to $\mathcal{NRL}(eq)$ to uniformly generate most aggregate functions found in practical query languages. We augment it with rational arithmetic ($+, -, *$ and $\div$) and a general summation operator, to obtain the language $\mathcal{NRL}^{\text{aggr}}$. We also consider a weaker language $\mathcal{NRL}^{\text{nat}}$ obtained from $\mathcal{NRL}^{\text{aggr}}$ by restricting the type of rationals to natural numbers only. We show that these language are closely related to $\mathcal{BQL}$.

Theorem 4.4 *The languages $\mathcal{BQL}$ and $\mathcal{NRL}^{\text{nat}}$ have the same expressive power.*

In Section 5, we study the expressive power of bag languages and set languages with aggregate functions. First, we prove that $\mathcal{NRL}^{\text{nat}}$ and $\mathcal{NRL}^{\text{aggr}}$ have the conservative extension property. That is, the expressibility of queries in these languages is independent of the height of nesting allowed in intermediate data.

To study limitations of the languages, we introduce the *bounded degree property* of a graph query q that says the following: if G is a graph whose in- and out-degrees do not exceed k, then there exists a number $c(q, k)$ (depending on q and k, but not G) such that the graph $q(G)$ has at most $c(q, k)$ distinct in- and out-degrees.

We demonstrate that using the bounded degree property it is easy to show the inexpressibility of a number of queries such as testing for a chain or testing for a balanced binary tree. It is also easy to use this property to show that *classes* of queries described by their behavior on certain types of graphs are not definable. The bounded degree is a property of the same kind as Gaifman's locality [19], and we are able to show that

Theorem 5.13 *Any first-order definable graph query has the bounded degree property.*

Using this result and conservativity, we obtain that $\mathcal{NRL}$-definable graph queries have the bounded degree property if the nodes are of base types. We also conjecture that this continues to hold in $\mathcal{NRL}^{\text{aggr}}$, but so far we have been unable to prove this. To settle the main conjectures for $\mathcal{NRL}^{\text{aggr}}$ (and hence for $\mathcal{BQL}$) we use a different technique. We define *k-multi-cycles* as graphs that consist of $n \geq 1$ connected components, each being a simple cycle of the same length $\geq k$. We prove a special case of the locality theorem for $\mathcal{NRL}^{\text{aggr}}$.

Theorem 5.15 *Let q be a $\mathcal{NRL}^{\text{aggr}}$-definable Boolean query on graphs whose nodes are of unordered base type. Then there exists a number k such that the value of q is the same for all k-multi-cycles.*

It follows from this theorem that the parity test and transitive closure are not definable in $\mathcal{NRL}^{\text{aggr}}$ and $\mathcal{BQL}$. We also formulate an analog of Theorem 5.15 for a different class of graphs, and from that result we conclude that the balanced binary tree test is not $\mathcal{BQL}$-definable.

In Section 6, we consider three extensions of $\mathcal{BQL}$: with power operators such as *powerset* [21], with structural recursion [6], and with loops [35, 48].

Theorems 6.3 and 6.4 *$\mathcal{BQL}$ with the powerset primitive is strictly less expressive than $\mathcal{BQL}$ with structural recursion, which in turn has the same expressive power as $\mathcal{BQL}$ with loops.*

We study the impact of these primitives on the arithmetic expressive power and prove the following.

Theorems 6.5 and 6.6 *The classes of arithmetic functions expressible in $\mathcal{BQL}$ enhanced with powerset and with loops are the classes of Kalmar elementary and primitive recursive functions respectively.*

Finally, we reexamine the equivalence $\mathcal{NRL}^{\text{nat}} \simeq \mathcal{BQL}$ in the presence of these operators. We show that $\mathcal{NRL}^{\text{nat}}$ enhanced with *powerset* or loops is strictly less expressive than $\mathcal{BQL}$ enhanced with the same primitives, and identify the gap between these languages. We define the new primitive *gen* that takes a number n as an input and produces the set $\{0, \ldots, n\}$, and prove that $\mathcal{BQL}$ with the *powerset* primitive has the same expressive power as $\mathcal{NRL}^{\text{nat}}$ with *powerset* and *gen*. The same holds for structural recursion and loops.

2 An Approach to Language Design

Many query languages have traditionally been developed on the basis of the relational algebra or the relational calculus. This approach may be too limiting when we have to deal with structures that are not naturally supported by first-order logic-based languages. This is certainly the case with nested structures or structures such as bags and lists, where multiplicity or the order of appearance is important.

Instead of using first-order logic as a universal platform, it was suggested by Cardelli [11] to consider the main type constructors (such as sets, bags, and records) independently and for each of them determine the introduction and elimination operations. The introduction operations allow us to construct the elements of a given type. The elimination operations compute with them.

This idea was further developed in Tannen et al. [8] and Buneman et al. [9], where it was suggested that one use operations *naturally* associated with data types as introduction and elimination operations. The formal category-theoretic treatment was given in Libkin [32]. The operations on collections correspond to the categorical notion of a monad [40].

We illustrate the use of this approach in the design of the nested relational language $\mathcal{NRL}$. This language deals with complex objects in the form of nested relations. The types of complex objects are given by the grammar below.

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

The semantics of types is as follow. The type *unit* is a special base type containing exactly the distinguished value denoted by (). The symbol b ranges over an unspecified collection of base types, such as integers, booleans, etc. Elements of the product type $s \times t$ are pairs whose first component is of type s and whose second component is of type t. Note that in this paper we use pairs instead of records. This does not affect expressiveness, but makes notation more manageable. Finally, elements of the set type $\{s\}$ are finite sets whose elements are of type s.

The only operation naturally associated with type *unit* is the one that produces the unique element of that type on any input. The introduction operation for pairs is pair formation: given x and y, form the pair (x, y). The elimination operations are first and second projections: $\pi_1\ (x, y) = x$ and $\pi_2\ (x, y) = y$.

For sets, the situation is not as straightforward because there are two ways to construct sets: either by starting with an empty set and inserting elements, or by starting with empty and singleton sets and using the union operation. We adopt the latter here. That is, each set is either $\emptyset$, or it is a singleton $\{x\}$ or it is the union of two sets $X \cup Y$. Assuming that $\emptyset$, singleton formation, and $\cup$ are the introduction operations, we define the elimination operation by prescribing its action in each of the three cases:

$$\begin{array}{lll} \mathsf{fun} & s\_sru(e, f, u)(\emptyset) & = & e \\ \mid & s\_sru(e, f, u)(\{x\}) & = & f(x) \\ \mid & s\_sru(e, f, u)(X \cup Y) & = & u(s\_sru(e, f, u)(X), s\_sru(e, f, u)(Y)) \end{array}$$

Following Tannen et al. [6, 7], *s_sru* stands for "structural recursion on the union presentation of sets." It has three parameters, e, f, and u. Setting these parameters arbitrarily leads to ill-defined programs. It is well known that checking whether a program using *s_sru* is well-defined is undecidable [7]. Hence, it was proposed [8] that some syntactic restrictions be imposed on *s_sru* to ensure well-definedness. In particular, this is achieved by taking e to be $\emptyset$ and u to be union. Then, the resulting function $s\_sru(\emptyset, f, \cup)$, which is called $s\_ext(f)$, has type $\{s\} \to \{t\}$ if f has type $s \to \{t\}$. Its semantics is given by

$$s\_ext(f)\ \{x_1, \ldots, x_n\} \;=\; f(x_1) \cup \ldots \cup f(x_n)$$

That is, it extends f to sets. Instead of $s\_ext(f)$, one can use the following two constructs. The s_μ operator flattens a set of sets: $s\_\mu\ \{X_1, \ldots, X_n\} = X_1 \cup \ldots \cup X_n$. The $s\_map(g)$ operator applies the function g to every element of a set: $s\_map(g)\ \{x_1, \ldots, x_n\} = \{g(x_1), \ldots, g(x_n)\}$. It is not hard to see [8] that $s\_ext(f) = s\_\mu \circ s\_map(\lambda x.f(x))$, $s\_\mu = s\_ext(id)$, and $s\_map(g) = s\_ext(\lambda x.\{g(x)\})$.

Using these operations, we present the nested relational language $\mathcal{NRL}$ here. It has three equally expressive components that can be freely combined: the nested relational algebra $\mathcal{NRA}$, based on the operations discussed above; the nested relational calculus $\mathcal{NRC}$; and relative set abstraction $\mathcal{RSA}$.

Expressions of $\mathcal{NRA}$, $\mathcal{NRC}$, and $\mathcal{RSA}$ are constructed using the rules in Figure 1. These rules give the most general types of the expressions. In the figure, we also show these types in the superscript. The type superscripts are omitted in subsequent sections as they can be inferred [26, 42, etc.] These constructs have been fully explained by Tannen et al. [8]. We briefly repeat their semantics here.

- 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, $(g \circ h)(d) = g(h(d))$.

EXPRESSIONS OF $\mathcal{NRA}$

General Operators

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

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

Set Operators

$$\frac{}{s\_\eta^s : s \to \{s\}} \qquad \frac{}{s\_\mu^s : \{\{s\}\} \to \{s\}} \qquad \frac{f : s \to t}{s\_map(f) : \{s\} \to \{t\}}$$

$$\frac{}{K\{\}^s : unit \to \{s\}} \qquad \frac{}{\cup^s : \{s\} \times \{s\} \to \{s\}} \qquad \frac{}{s\_\rho_2^{s,t} : s \times \{t\} \to \{s \times t\}}$$

EXPRESSIONS OF $\mathcal{NRC}$

Lambda Calculus and Products

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

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

Set Expressions

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

EXPRESSIONS OF $\mathcal{RSA}$

All constructs of $\mathcal{NRC}$, except $\bigcup\{e_1 \mid x \in e_2\}$, and

Relative Set Abstraction Construct

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

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

- 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 set.
- $\cup$ is set union.
- s_η forms singleton sets; for example, $s\_\eta\ 3$ evaluates to $\{3\}$.
- s_μ flattens a set of sets; for example, $s\_\mu\{\{1,2,3\},\{1,3,5,7\},\{2,4\}\}$ evaluates to $\{1,2,3,4,5,7\}$.
- $s\_map(f)$ applies f to every item in the input set; for example $s\_map(\lambda x.1+x)\{1,2,3\}$ yields $\{2,3,4\}$ and $s\_map(\lambda x.1)\{1,2,3\}$ yields $\{1\}$.
- $s\_\rho_2(x, y)$ pairs x with every item in the set y; for example, $s\_\rho_2(1,\{1,2\})$ returns $\{(1,1),(1,2)\}$.
- $\bigcup\{e_1 \mid x \in e_2\}$ is equivalent to $s\_ext(\lambda x.e_1)(e_2)$, that is, $(s\_\mu \circ s\_map(\lambda x.e_1))(e_2)$. For example, $\bigcup\{s\_\eta(x-1) \cup s\_\eta(x+1) \mid x \in \{2,4\}\}$ evaluates to $\{1,3,5\}$.
- $\{e \mid x_1 \in e_1, ..., x_n \in e_n\}$ is equivalent to $\bigcup\{\ldots\bigcup\{\{e\} \mid x_n \in e_n\}\ldots \mid x_1 \in e_1\}$. It can be understood as a normal comprehension notation from set theory; see Wadler [58] and Buneman et al. [9]. For instance, $\{(x, y) \mid x \in X, y \in Y\}$ is the Cartesian product of sets X and Y.

The whole of $\mathcal{NRL}$ is used in many places of this report. However, in many of the proofs only one of $\mathcal{NRA}$, $\mathcal{NRC}$, or $\mathcal{RSA}$ is used. This is fine because these three sublanguages are equivalent in terms of denotations and in terms of equational theories [8, 59].

Proposition 2.1 *$\mathcal{NRA}$, $\mathcal{NRC}$, and $\mathcal{RSA}$ are equivalent in terms of semantics. In fact, the translations between them preserve and reflect their respective equational theories.* □

Tannen et al. [8] represented booleans by the two values of type $\{unit\}$, that is, $\{()\}$ for true and $\{\}$ for false. It was shown [8] that after adding an equality test primitive $eq^s : s \times s \to \{unit\}$ for each complex object type s, $\mathcal{NRL}$ expresses all nested relational operations of the well-known algebra of Thomas and Fischer [54]. In fact, this result can be strengthened [60] because the converse is also true if a few constant relations are added to the algebra of Thomas and Fischer [54], which is known to be equivalent to the language of Colby [13] and to the language of Schek and Scholl [49]. Also [60], real booleans can be added to $\mathcal{NRL}$ as a base type together with equality tests $=^s$: $s \times s \to bool$ and the conditional construct to yield a language that has the same strength as $\mathcal{NRL}(eq)$ (we list the additional primitives explicitly in brackets to distinguish the various versions of $\mathcal{NRL}$). Consequently, we have

Proposition 2.2 $\mathcal{NRL}(eq) \simeq \mathcal{NRL}(=, bool, cond) \simeq Thomas\&Fischer \simeq Schek\&Scholl \simeq Colby$. □

Here and in what follows $\mathcal{L}_1 \simeq \mathcal{L}_2$ means that $\mathcal{L}_1$ and $\mathcal{L}_2$ have the same expressive power. Occasionally we use this notation when $\mathcal{L}_1$ and $\mathcal{L}_2$ have different type systems. This only happens when there exist translations between the type systems and the equivalence is meant to be the equivalence of expressive power with respect to those translations. We shall also use $\mathcal{L}_1 \subseteq \mathcal{L}_2$ when $\mathcal{L}_2$ is at least as expressive as $\mathcal{L}_1$, and $\mathcal{L}_1 \subset \mathcal{L}_2$ when $\mathcal{L}_2$ is strictly more expressive than $\mathcal{L}_1$.

For the sake of clarity, pattern matching is used in many places later on in this report. It can be removed in a straightforward manner. For example, $\lambda X.\{(a, \{b \mid (c,b) \in X, c = a\}) \mid (a,z) \in X\}$ is just a syntactic sugar for $\lambda X.\{(\pi_1\ x,\ \{\pi_2\ y \mid y \in X, w \in (\pi_1\ y\ eq\ \pi_1\ x)\}) \mid x \in X\}$, which is the function implementing the nest operation of the nested relational algebra.

3 The Basic Bag Language $\mathcal{BQL}$

3.1 The Ambient Bag Language $\mathcal{NBL}$

Using the approach outlined in the previous section, we now define an ambient bag query language $\mathcal{NBL}$ consisting of three corresponding components: the bag algebra $\mathcal{NBA}$, the bag calculus $\mathcal{NBC}$, and the relative bag abstraction $\mathcal{RBA}$. We use the $\{\!|\cdot,\ldots,\cdot|\!\}$ brackets for bags.

Similar to sets, bags can be constructed from empty bag $\{\!|\,|\!\}$ and singleton bags $\{\!|x|\!\}$ using the *additive union* operation $\uplus$ that adds up multiplicities. The general elimination operation, called structural recursion on union presentation of bags, *b_sru*, is defined by prescribing its action depending on the structure of a bag:

$$
\begin{array}{lllll}
\mathsf{fun} & b\_sru(e,f,u)(\{\!|\,|\!\}) & = & e \\
\mid & b\_sru(e,f,u)(\{\!|x|\!\}) & = & f(x) \\
\mid & b\_sru(e,f,u)(B_1 \cup B_2) & = & u(b\_sru(e,f,u)(B_1), b\_sru(e,f,u)(B_2))
\end{array}
$$

Similar to sets, there are choices of e, f, and u that make $b\_sru(e,f,u)$ ill-defined and well-definedness is again undecidable [7]. So we ensure well-definedness by imposing syntactic restrictions on *b_sru*: e is required to be $\{\!|\,|\!\}$ and u is required to be $\uplus$, leaving f the only parameter that may vary. The resulting construct, denoted by $b\_ext(f)$, has the semantics below and is equivalent to mapping over a bag followed by flattening a bag of bags.

$$b\_ext(f)\ \{\!|x_1,\ldots,x_n|\!\} = f(x_1) \uplus \ldots \uplus f(x_n)$$

Summing up, we now have an ambient bag language which we called $\mathcal{NBL}$. The expressions of $\mathcal{NBL}$ are given in Figure 2. The types of $\mathcal{NBL}$ are the same as $\mathcal{NRL}$ but uses bags instead of sets. That is,

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

where elements of type $\{\!|s|\!\}$ 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.

The semantics of $\mathcal{NBL}$ is similar to the semantics of $\mathcal{NRL}$. The difference is in the operations such as $\uplus$ and b_μ that add up multiplicities. b_η forms singleton bags; for example, $b\_\eta\ 3$ evaluates to the singleton bag $\{\!|3|\!\}$. b_μ flattens a bag of bags; for example, $b\_\mu\ \{\!|\{\!|1,2,3|\!\},\{\!|1,3,5,7|\!\},\{\!|2,4|\!\}|\!\}$ evaluates to $\{\!|1,2,3,1,3,5,7,2,4|\!\}$. $b\_map(f)$ applies f to every item in the input bag; for example, $b\_map(\lambda x.1+$

EXPRESSIONS OF $\mathcal{NBA}$

Operations of category with products as in $\mathcal{NRA}$.

Bag Operators

$$\frac{}{b\_\eta^s : s \to \{\!|s|\!\}} \qquad \frac{}{b\_\mu^s : \{\!|\{\!|s|\!\}|\!\} \to \{\!|s|\!\}} \qquad \frac{f : s \to t}{b\_map(f) : \{\!|s|\!\} \to \{\!|t|\!\}}$$

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

EXPRESSIONS OF $\mathcal{NBC}$

Operations of lambda calculus and products as in $\mathcal{NRC}$.

Bag Operators

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

EXPRESSIONS of $\mathcal{RBA}$

All operations of $\mathcal{NBC}$, except $\biguplus \{\!|e_1 \mid x^s \in e_2|\!\}$, and

Relative Bag Abstraction Construct

$$\frac{e : s \quad e_1 : \{\!|s_1|\!\} \quad \ldots \quad e_n : \{\!|s_n|\!\}}{\{\!|e \mid x_1^{s_1} \in e_1, \ldots, x_n^{s_n} \in e_n|\!\}}$$

Figure 2: Expressions of $\mathcal{NBL}$

$x)$ $\{\!|1,2,1,6|\!\}$ evaluates to $\{\!|2,3,2,7|\!\}$ and $b\_map(\lambda x.1)$ $\{\!|1,2,1,6|\!\}$ evaluates to $\{\!|1,1,1,1|\!\}$. $K\{\!||\!\}$ forms empty bags of the appropriate types. $\uplus$ is additive union of bags; for example, $\uplus(\{\!|1,2,3|\!\},\{\!|2,2,4|\!\})$ returns $\{\!|1,2,3,2,2,4|\!\}$. b_ρ_2 pairs the first component of the input with every item in the second component of the input; for example, $b\_\rho_2(3,\{\!|1,2,3,1|\!\})$ returns $\{\!|(3,1),(3,2),(3,3),(3,1)|\!\}$. The meaning of $\biguplus\{\!|e_1 \mid x^s \in e_2|\!\}$ is to flat-map the function $\lambda x.e_1$ over the bag e_2. That is, $\biguplus\{\!|e_1 \mid x \in e_2|\!\}$ is equivalent to $(b\_\mu \circ b\_map(\lambda x.e_1))(e_2)$. The semantics of $\{\!|e \mid x_1 \in e_1,\ldots,x_n \in e_n|\!\}$ is just $\biguplus\{\!|\ldots\biguplus\{\!|\{\!|e|\!\} \mid x_n \in e_n|\!\}\ldots \mid x_1 \in e_1|\!\}$. It is a most convenient and easy to understand construct. For example, $\{\!|(x,y) \mid x \in e_1, y \in e_2|\!\}$ is just the "cartesian product" of bags e_1 and e_2.

Similar to $\mathcal{NRL}$, the three components of $\mathcal{NBL}$ are equally expressive. In fact, the proof is identical to that used for $\mathcal{NRL}$ [8].

Proposition 3.1 *$\mathcal{NBA}$, $\mathcal{NBC}$, and $\mathcal{RBA}$ are equivalent in terms of denotations. Moreover, the translations between them preserve and reflect their equational theories.* □

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

3.2 Relative Strength of Bag Operators and the Language $\mathcal{BQL}$

As mentioned earlier, 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, where we have well established yardsticks, very little is known for bags. Due to this lack of adequate guideline, a large number of primitives are considered. These primitives are either "invented" by us or are reported by other researchers, especially Albert [4] and Grumbach and Milo [21]. In contrast to Grumbach and Milo [21] who included a *powerbag* operator as a primitive, all operators considered by us have polynomial time complexity. We give a complete report of their expressive strength relative to the ambient bag language.

Let us first fix some meta notations. We define $count(d,B)$ to be the number of times the object d occurs in the bag B. To define the semantics of a bag operation $e(B_1,\ldots,B_n)$, it suffices to express $count(d,e(B_1,\ldots,B_n))$ in terms of $count(d,B_1)$, ..., $count(d,B_n)$. The bag operations to be considered are listed below.

$$\begin{array}{rcll}
monus & : & \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|s|\!\} & \text{bag difference} \\
max & : & \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|s|\!\} & \text{maximum union} \\
min & : & \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|s|\!\} & \text{minimum intersection} \\
eq & : & s \times s \to \{\!|unit|\!\} & \text{equality test} \\
member & : & s \times \{\!|s|\!\} \to \{\!|unit|\!\} & \text{membership test} \\
subbag & : & \{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|unit|\!\} & \text{subbag test} \\
unique & : & \{\!|s|\!\} \to \{\!|s|\!\} & \text{duplicate elimination}
\end{array}$$

The semantics of these additional operations is given in Figure 3. Notice that we are simulating booleans using a bag of type $\{\!|unit|\!\}$. True is represented by the singleton bag $\{\!|()|\!\}$ and False is represented by the empty bag $\{\!||\!\}$.

$$
\begin{aligned}
count(d, monus(B_1, B_2)) &= \max(count(d, B_1) - count(d, B_2), 0) \\
count(d, max(B_1, B_2)) &= \max(count(d, B_1), count(d, B_2)) \\
count(d, min(B_1, B_2)) &= \min(count(d, B_1), count(d, B_2)) \\
eq(d_1, d_2) &= \begin{cases} \{\!|()|\!\} & \text{if } d_1 = d_2 \\ \{\!||\!\} & \text{otherwise} \end{cases} \\
member(d, B) &= \begin{cases} \{\!|()|\!\} & \text{if } count(d, B) > 0 \\ \{\!||\!\} & \text{otherwise} \end{cases} \\
subbag(B_1, B_2) &= \begin{cases} \{\!|()|\!\} & \text{if } count(d, B_1) \leq count(d, B_2) \text{ for every } d \\ \{\!||\!\} & \text{otherwise} \end{cases} \\
count(d, unique(B)) &= \begin{cases} 1 & \text{if } count(d, B) > 0 \\ 0 & \text{if } count(d, B) = 0 \end{cases}
\end{aligned}
$$

Figure 3: Semantics of additional bag operations

As emphasized in the introduction, each of these operators has polynomial time complexity with respect to the size of the input. Hence

Proposition 3.2 *Every function definable in $\mathcal{NBL}(monus, max, min, eq, member, unique)$ has polynomial time complexity with respect to the size of the input.* □

In the remainder of this section, the expressive power of these primitives is compared. The result is the following complete characterization of their relative expressive power.

Theorem 3.3 *monus can express all primitives other than unique which is independent from 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 from max.*

The results of this theorem can be visualized in the following diagram:

$$
\begin{tikzcd}[arrows=-]
 & monus \arrow[d] & & & \\
 & min \arrow[r, phantom, "\equiv"] \arrow[dl] \arrow[d] & subbag & & unique \\
max & eq \arrow[r, phantom, "\equiv"] & member & &
\end{tikzcd}
$$

As a consequence of these results, $\mathcal{NBL}(monus, unique)$ can be considered as the most powerful candidate for a standard bag query language. Thus, we propose to use it as the standard query language for bags.

Definition. *The Bag Query Language, $\mathcal{BQL}$, is defined as $\mathcal{NBL}$ endowed with monus and unique.*

In the remainder of the section we prove Theorem 3.3. Let us first prove the easy expressibility results. After that, the harder inexpressibility results are presented.

Proposition 3.4

- *max can be expressed in $\mathcal{NBL}(monus)$.*
- *min can be expressed in $\mathcal{NBL}(monus)$.*
- *eq can be expressed in $\mathcal{NBL}(monus)$.*
- *subbag can be expressed in $\mathcal{NBL}(monus)$.*
- *subbag can be expressed in $\mathcal{NBL}(eq, max)$.*
- *member can be expressed in $\mathcal{NBL}(eq)$.*
- *eq can be expressed in $\mathcal{NBL}(member)$.*
- *eq can be expressed in $\mathcal{NBL}(min)$.*
- *subbag can be expressed in $\mathcal{NBL}(min)$.*
- *min can be expressed in $\mathcal{NBL}(subbag)$.*
- *max can be expressed in $\mathcal{NBL}(min)$.*

Proof. To reduce clutter, we use the primitives in infix form.

- $B_1 \ max \ B_2 := B_2 \uplus (B_1 \ monus \ B_2)$.
- $B_1 \ min \ B_2 := B_1 monus (B_1 monus B_2)$.
- $d_1 \ eq \ d_2 := \{\!|()|\!\} \ monus \ (R_{12} \uplus R_{21})$ where R_{ij} is $\bigcup \{\!| \{\!|()|\!\} \mid x \in \{\!|d_i|\!\} \ monus \ \{\!|d_j|\!\} |\!\}$.
- $B_1 \ subbag \ B_2 := B_1 \ eq \ (B_1 monus (B_1 monus B_2))$.
- $B_1 \ subbag \ B_2 := B_2 \ eq \ (B_1 \ max \ B_2)$.
- $d \ member \ B := (\{\!|() \mid x \in B, y \in (x \ eq \ d)|\!\} \ eq \ \{\!||\!\}) \ eq \ \{\!||\!\}$.
- $d_1 \ eq \ d_2 := d_1 \ member \ \{\!|d_2|\!\}$.
- $d_1 \ eq \ d_2 := \{\!|() \mid x \in \{\!|d_1|\!\} \ min \ \{\!|d_2|\!\}|\!\}$.
- $B_1 \ subbag \ B_2 := B_1 \ eq \ (B_1 \ min \ B_2)$.
- $B_1 \ min \ B_2 := E \ \uplus \ F_{12} \ \uplus \ F_{21}$, where E is the bag of elements having the same number of occurrences in B_1 and B_2, and F_{ij} is the bag of elements of B_i that occur strictly less frequently in B_i than in B_j.

  Let X *intersection* Y be the function that returns those elements that occur the same number of times in X and Y. Let X *difference* Y be the function that returns those elements in X that occur different number of times in X and Y. Then E can be defined as B_1 *intersection* B_2, and F_{ij} as $\{\!|x \mid x \in B_i \ difference \ B_j, \ z \in \{\!|y \mid y \in B_i, \ w \in y \ eq \ x|\!\} \ subbag \ \{\!|y \mid y \in B_j, w \in y \ eq \ x|\!\}|\!\}$.

It remains to define *intersection* and *difference*. First observe that d_1 eq $d_2 := \{\!|() \mid x \in \{\!|d_1|\!\}$ $subbag$ $\{\!|d_2|\!\}, y \in \{\!|d_2|\!\}$ $subbag$ $\{\!|d_1|\!\}|\!\}$. Now B_1 *intersection* $B_2 := \{\!|x| \; x \in B_1, \; w \in \{\!|y \mid y \in B_1, \; z \in y \; eq \; x|\!\} \; eq \; \{\!|y \mid y \in B_2, \; z \in y \; eq \; x|\!\}|\!\}$. Finally, B_1 *difference* $B_2 := \{\!|x \mid x \in B_1, w \in (x \; member \; (B_1 \; intersection \; B_2)) eq \; \{\!||\!\}|\!\}$. Incidentally, it is also easy to show that *eq*, *intersection*, *difference*, and *member* are inter-expressible.

- B_1 *max* $B_2 := E \uplus F_{12} \uplus F_{21}$, where E is the bag of those elements that occur equally frequently in B_1 and B_2, and F_{ij} is the bag containing those elements of B_j that occur strictly more frequently in B_j than in B_i. Thus, E can be defined as B_1 *intersection* B_2 and F_{ij} as $\{\!|x \mid x \in B_j \; difference \; B_i, \; w \in \{\!|y \mid y \in B_i, z \in y \; eq \; x|\!\} \; subbag \; \{\!|y \mid y \in B_j, z \in y \; eq \; x|\!\}|\!\}$. □

In contrast to $\mathcal{NRL}$, where the primitives *eq*, *subset*, $-$, $\cap$ and *member* are interdefinable [8], the corresponding bag primitives differ considerably in expressive power. These inexpressibility results require arguments that are more cunning. We prove them in separate propositions below.

Proposition 3.5 *eq cannot be expressed in* $\mathcal{NBL}(unique, max)$.

Proof. Define the relation $\sqsubseteq_t$ on complex objects of type t by induction as follows: $d_1 \sqsubseteq_b d_2$ iff $d_1 = d_2$; $(d_1, d_2) \sqsubseteq_{s \times t} (d_1', d_2')$ if $d_1 \sqsubseteq_s d_1'$ and $d_2 \sqsubseteq_t d_2'$; $B_1 \sqsubseteq_{\{\!|s|\!\}} B_2$ if for every d_1 such that $count(d_1, B_1) \neq 0$, there is some d_2 such that $count(d_2, B_2) \neq 0$ and $d_1 \sqsubseteq_s d_2$. It is not difficult to check that every function definable in $\mathcal{NBA}(unique, max)$ is monotone with respect to $\sqsubseteq$. However, *eq* is not monotone with respect to $\sqsubseteq$. □

Proposition 3.6 *unique cannot be expressed in* $\mathcal{NBL}(monus)$.

Proof. The technique of Wong [59] can be readily adapted to show that the rewrite system below is strongly normalizing.

- $(\lambda x.e)(e') \leadsto e[e'/x]$
- $\pi_i(e_1, e_2) \leadsto e_i$
- $\{\!|e \mid \Delta_1, x \in \{\!||\!\}, \Delta_2|\!\} \leadsto \{\!||\!\}$
- $\{\!|e \mid \Delta_1, x \in \{\!|e'|\!\}, \Delta_2|\!\} \leadsto \{\!|e[e'/x] \mid \Delta_1, \Delta_2[e'/x]|\!\}$
- $\{\!|e \mid \Delta_1, x \in e_1 \uplus e_2, \Delta_2|\!\} \leadsto A_1 \uplus A_2$, where A_i is $\{\!|e \mid \Delta_1, x \in e_i, \Delta_2|\!\}$.
- $\{\!|e \mid \Delta_1, x \in \{\!|e' \mid \Delta'|\!\}, \Delta_2|\!\} \leadsto \{\!|e[e'/x] \mid \Delta_1, \Delta', \Delta_2[e'/x]|\!\}$
- $(e_1 \; monus \; e_2) \leadsto e$, where e_1, e_2 have no free variables and e is the result of evaluating $e_1 \; monus \; e_2$.

It can be shown that the rewrite system obtained by adjoining the rule below to the above system is weakly normalizing:

- $\{\!|e \mid \Delta_1, x \in e_1 \; monus \; e_2, \Delta_2|\!\} \leadsto \{\!|\pi_2 \; y \mid y \in A_1 \; monus \; A_2|\!\}$, where A_i is $\{\!|(x, e) \mid \Delta_1, x \in e_i, \Delta_2|\!\}$ and at least one of Δ_j is not empty.

Now we argue that no normal form under these rules implements *unique*. Let b be a new base type. Let $O : \{\!|b|\!\}$ be a bag having $k > 1$ identical elements and nothing else. It can be checked that any normal form implementing *unique* must have the form $\lambda R.e$, where R is the only free variable of e. Then the proposition follows from the claim below.

Claim. Let $A : \{\!|\ldots \times b \times \ldots|\!\}$ be a normal form having $R : \{\!|b|\!\}$ as its only free variable. Then for any $o : \ldots \times b \times \ldots$, $count(o, A[O/R])$ is a multiple of k.

Proof of claim. We proceed by analyzing the forms that A can take. When A is R, the count is $1 \cdot k$ if o occurs in R or is $0 \cdot k$ otherwise. When A is $\{\!||\!\}$, the count is $0 \cdot k$. When A is $B \uplus C$ or B *monus* C, the hypothesis is applicable to both B and C. So $count(o, B[O/R]) = m_B \cdot k$ and $count(o, C[O/R]) = m_C \cdot k$ respectively. Then $count(o, A[O/R])$ is $(m_B + m_C) \cdot k$ or $(m_B \dot{-} m_C) \cdot k$ respectively.

There are two remaining possibilities. The first possibility is for A to take the form $\{\!|e \mid x_1 \in R, \ldots, x_m \in R|\!\}$. In this case, the count of any o in $A[O/R]$ is obviously a multiple of k. The second possibility is for A to take the form $\{\!|e \mid x \in B \textit{ monus } C|\!\}$. In this case, e must have type $\ldots \times b \times \ldots$. It can be checked that this forces x to have a type that is not necessarily the same as e's but is still of the form $\ldots \times b \times \ldots$. As a result, B *monus* C is a normal form whose type has the form $\{\!|\ldots \times b \times \ldots|\!\}$. So the hypothesis can be applied and the count of an element in B *monus* C must be a multiple of k. Let o be an element of $A[O/R]$. Let o_1, ..., o_n be the distinct elements of $(B \textit{ monus } C)[O/R]$ such that $e[o_1/x]$, ..., $e[o_n/x]$ all evaluate to o. Then $count(o, A[O/R]) = (m_1 + \ldots + m_n) \cdot k$, where m_i's are the count of o_i's in $(B \textit{ monus } C)[O/R]$. $\square$

Proposition 3.7 *monus cannot be expressed in* $\mathcal{NBL}(subbag)$.

Proof. Let e be an expression of $\mathcal{NBC}(subbag)$ in normal form (induced by the rewrite system of the previous proposition) having no constants of base type b and no function abstraction. Let its free variables be $x_1 : t_1$, ..., $x_n : t_n$. Let θ assign object $\theta(x_i)$ of type t_i to x_i. Let b_1, ..., b_m be all the bags of type $\{\!|b|\!\}$ appearing in $\theta(x_1)$, ..., $\theta(x_n)$. Let a_1, ..., a_l be all the objects of type b in $\theta(x_1)$, ..., $\theta(x_n)$. Associate with each a_i a set $\kappa a_i = \{q_0, \ldots, q_m\}$, where $q_0 = 1$ if an occurrence of a_i in some $\theta(x_j)$ is not inside some of b_1, ..., b_m; $q_0 = 0$ otherwise; $q_{1 \leq j \leq m} =$ the number of times a_i appears in b_j. Let $e\theta$ evaluate to an object o. By structural induction on e, the number of occurrences of a_i in o can be expressed by a formula of the form: $p_0 \cdot q_0 + \ldots + p_m \cdot q_m$ where $\kappa a_i = \{q_0, \ldots, q_m\}$ and p_0, ..., p_m are natural numbers. However, *monus* clearly does not have this property. $\square$

Proposition 3.8 *Let* $MIN : \{\!|\{\!|s|\!\}|\!\} \to \{\!|s|\!\}$ *be the function such that for every* d,

$$count(d, MIN(R)) = \min\{count(d, X) \mid X \in R\}.$$

Then

- *MIN cannot be expressed in* $\mathcal{NBL}(monus)$.
- *MIN cannot be expressed in* $\mathcal{NBL}(unique, member)$.
- *MIN can be expressed in* $\mathcal{NBL}(unique, member, max)$.
- *subbag cannot be expressed by* $\mathcal{NBL}(member)$.
- *max cannot be expressed by* $\mathcal{NBL}(unique, member)$.

Proof. The last two items are immediate consequences of the first three items, which we prove below.

- Since *unique* cannot be expressed in $\mathcal{NBL}(monus)$, it suffices to show that it is expressible in $\mathcal{NBL}(MIN, eq)$. Clearly, $unique(B) := MIN\{\!|\{\!|r|\!\} \uplus \{\!|x \mid x \in B, y \in (x\ eq\ r)\ eq\ \{\!||\!\}|\!\} \mid r \in B|\!\}$.

- From Section 4, it is not difficult to see that $\mathcal{NBL}(unique, member) \simeq \mathcal{NRL}(\mathbb{N}, \Sigma, \cdot, +, eq)$ and $\mathcal{NBL}(unique, subbag) \simeq \mathcal{NRL}(\mathbb{N}, \Sigma, \cdot, +, eq, \leq)$, where we add natural numbers, some limited arithmetic, and a summation primitive to $\mathcal{NRL}$. Clearly, $\mathcal{NRL}(\mathbb{Q},\ \Sigma, \cdot, +, \div, bool,\ cond,\ =) \supseteq \mathcal{NRL}(\mathbb{N}, \Sigma, \cdot, +, eq)$. It is proved in Section 5 that $\leq: \mathbb{N} \times \mathbb{N} \to bool$ is not expressible in the former nested relational language. Hence it cannot be expressed in the latter. Consequently, $\mathcal{NBL}(unique, member)$ cannot express *subbag*. But $\mathcal{NBL}(MIN)$ easily expresses *subbag*. So *MIN* cannot be expressible in $\mathcal{NBL}(unique, member)$.

- First note that *subbag* can be expressed in $\mathcal{NBL}(member, max)$. Clearly, *MIN* is expressible in $\mathcal{NBL}(unique, subbag)$ as $MIN(R) := (b\_\mu \ \circ \ unique)(B)$, where B is $\{\!|w \mid (y, w) \in A,$ $not\{\!|() \mid (x, v) \in A,\ x = y, w \neq v, v\ subbag\ w|\!\}|\!\}$ and A is $\{\!|(y, \{\!|x \mid x \in v, x = y|\!\}) \mid u \in R, y \in u, v \in R|\!\}$. □

This finishes the proof of Theorem 3.3. The independence of *unique* was also proved by Van den Bussche and Paredaens [56] and Grumbach and Milo [21], and the fact that *monus* is the strongest amongst the remaining primitives was also shown by Albert [4]. 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 3.3 which completely and strictly summarizes the relative strength of these primitives.

4 Relationship Between Bags and Sets

The relationship between sets and bags can be investigated from two perspectives. First, we compare several of the nested bag languages 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. Second, we consider augmenting $\mathcal{NRL}(eq)$ by new primitives with the aim of simulating $\mathcal{BQL}$, the most powerful of bag languages considered so far. In this way, we hope to understand the precise character of the new expressive power that bags bring us.

4.1 Set-Theoretic Expressive Power of Bag Languages

In order to compare bags and sets, two technical devices are required for conversions between bags and sets.

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

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 a the bag $\{\!|4,4,4|\!\}$. In particular, $sb\_map(f) = b\_map(f) \circ sb\_map(id)$. This axiom guarantees that when a set is converted to a bag using $sb\_map(id)$, the resulting bag contains no duplicates. Bags without duplicates are considered in this section as the most natural representation of sets using bags.

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

$$
\begin{array}{ll}
to\_bag_b := id & from\_bag_b := id \\
to\_bag_{s \times t} := \langle to\_bag_s \circ \pi_1, to\_bag_t \circ \pi_2 \rangle & from\_bag_{s \times t} := \langle from\_bag_s \circ \pi_1, from\_bag_t \circ \pi_2 \rangle \\
to\_bag_{\{s\}} := sb\_map(to\_bag_s) & from\_bag_{\{\!|s|\!\}} := bs\_map(from\_bag_s)
\end{array}
$$

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 $f' : to\_bag(s) \to to\_bag(t)$ definable in $\mathcal{NBL}(\Gamma)$ and the diagram below commutes.

$$
\begin{tikzcd}
to\_bag(s) \arrow[r, "f'"] & to\_bag(t) \arrow[r, "id"] & to\_bag(t) \arrow[d, "from\_bag_{to\_bag(t)}"] \\
s \arrow[u, "to\_bag_s"] \arrow[r, "f"'] & t \arrow[u, "to\_bag_t"'] \arrow[r, "id"'] & t
\end{tikzcd}
$$

The class $\mathcal{SET}(\Gamma)$ is precisely the class of "set-theoretic" functions expressible in $\mathcal{NBL}(\Gamma)$. That is, it corresponds to the class of functions from duplicate-free bags to duplicate-free bags in $\mathcal{NBL}(\Gamma)$. We compare $\mathcal{SET}(\Gamma)$ with $\mathcal{NRL}(eq)$ for various bag primitives below. Let eq_b be equality test restricted to base types. Let $empty : \{\!|unit|\!\} \to \{\!|unit|\!\}$ be a primitive such that it returns the bag $\{\!|()|\!\}$ when applied to the empty bag and returns the empty bag otherwise.

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

Proof. It is easy to check [59] that $\mathcal{NRL}(eq) = \mathcal{NRL}(eq_b, not)$ where $not : \{unit\} \to \{unit\}$ returns $\{()\}$ if applied to the empty set and returns $\{\}$ otherwise. Hence we prove $\mathcal{SET}(unique, eq_b, empty) = \mathcal{NRL}(eq_b, not)$ instead. To show $\mathcal{NRL}(eq_b, not) \subseteq \mathcal{SET}(unique, eq_b, empty)$, we prove that for any $f : s \to t$ in $\mathcal{NRA}(eq_b, not)$, there is $f' : to\_bag(s) \to to\_bag(t)$ in $\mathcal{NBA}(unique, eq_b, empty)$ such that the diagram below commutes.

$$
\begin{tikzcd}
to\_bag(s) \arrow[r, "f'"] & to\_bag(t) \arrow[r, "id"] & to\_bag(t) \arrow[d, "from\_bag_{to\_bag(t)}"] \\
s \arrow[u, "to\_bag_s"] \arrow[r, "f"'] & t \arrow[u, "to\_bag_t"'] \arrow[r, "id"'] & t
\end{tikzcd}
$$

First note that the right square in the above diagram obviously commutes. Therefore, we need only to prove that the left square commutes. This is straightforward by defining f' as follows:

$$
\begin{array}{llll}
id' := id & K\{\}' := K\{\!|\,|\!\} & \pi_1' := \pi_1 & \pi_2' := \pi_2 \\
Kc' := Kc & !' := \, ! & not' := empty & s\_\rho_2' := b\_\rho_2 \\
s\_\mu' := unique \circ b\_\mu & (s\_map\ g)' := unique \circ b\_map\ g' & (g \circ h)' := g' \circ h' & \langle g, h\rangle' := \langle g', h'\rangle \\
s\_\eta' := b\_\eta & eq_b' := eq_b & \cup' := unique \circ \uplus &
\end{array}
$$

The reverse inclusion $\mathcal{SET}(unique, eq_b, empty) \subseteq \mathcal{NRL}(eq_b, not)$ follows by showing that for any $f : s \to t$ in $\mathcal{NBA}(unique, eq_b, empty)$ there is an $f'' : from\_bag(s) \to from\_bag(t)$ in $\mathcal{NRL}(eq_b, not)$ such that

$$
\begin{tikzcd}
from\_bag(s) \arrow[r, "f''"] & from\_bag(t) \\
s \arrow[u, "from\_bag_s"] \arrow[r, "f"'] & t \arrow[u, "from\_bag_t"']
\end{tikzcd}
$$

This is straightforward by defining f'' as follows:

$$
\begin{array}{llll}
id'' := id & K\{\!|\,|\!\}'' := K\{\} & \pi_1'' := \pi_1 & \pi_2'' := \pi_2 \\
Kc'' := Kc & !'' := \, ! & empty'' := not & b\_\rho_2'' := s\_\rho_2 \\
b\_\mu'' := s\_\mu & (b\_map\ g)'' := s\_map\ g'' & (g \circ h)'' := g'' \circ h'' & \langle g, h\rangle'' := \langle g'', h''\rangle \\
b\_\eta'' := s\_\eta & eq_b'' := eq_b & unique'' := id & \uplus'' := \cup
\end{array}
$$

□

Note that the use of *unique* cannot be removed from the theorem above without weakening the notion of what kind of bags is considered an acceptable representation of sets. As mentioned earlier, only duplicate-free bags are accepted here as representation of sets. To see that *unique* cannot be removed, consider how one can define the equivalent of set union $\cup : \{s\} \times \{s\} \to \{s\}$ using bags. The bag-equivalent of set union must take a pair of duplicate-free bags to a duplicate-free bag having exactly those elements in the two input bags. The only binary operation from $\{\!|s|\!\} \times \{\!|s|\!\} \to \{\!|s|\!\}$ that combines data from both its input bags in the ambient bag language is $\uplus$. However, $\uplus$ takes two duplicate-free bags having nonempty intersection to a bag with duplicates. In other words, the output of $\uplus$ is not acceptable as a bag representing a set. Since duplicate elimination has been shown in the previous section to be independent of the other bag operators, we have no choice but apply *unique* to the result in order to obtain a bag that meets our standard.

Proposition 4.2 $\mathcal{NRL}(eq) \subset \mathcal{SET}(unique, eq)$

Proof. Since $\mathcal{NRL}(eq)$ is a conservative extension of the flat relational algebra [59], it cannot test whether two given sets have the same cardinality [28]. However, this function is defined in $\mathcal{SET}(unique, eq)$ as $from\_bag \circ eq \circ \langle b\_map\ ! \circ \pi_1, b\_map\ ! \circ \pi_2\rangle \circ to\_bag$. □

Proposition 4.3 *$\mathcal{NRL}(eq)$ and $\mathcal{SET}(monus)$ are incomparable.*

Proof. It is immediate that $\mathcal{SET}(monus) \not\subseteq \mathcal{NRL}(eq)$ because the function which tests if two sets have equal cardinality is in the former but not the latter. To show that $\mathcal{NRL}(eq) \not\subseteq \mathcal{SET}(monus)$, consider the relational projection $\Pi_1\{(x_1, y_1), \ldots, (x_n, y_n)\} = \{x_1, \ldots, x_n\}$. Π_1 is clearly in $\mathcal{NRL}(eq)$. Suppose it is also in $\mathcal{SET}(monus)$. Then the function $\Pi'_1\{\!|(o_1, o_1^1), \ldots, (o_1, o_1^{m_1}), \ldots, (o_n, o_n^1), \ldots, (o_n, o_n^{m_n})|\!\} = \{\!|o_1, \ldots, o_n|\!\}$, where o_1, ..., o_n are distinct, is definable in $\mathcal{NBL}(monus)$. Then $unique(R) = \Pi'_1(\{\!|(x, x) \mid x \in R|\!\})$ is definable in $\mathcal{NBL}(monus)$, contradicting Proposition 3.6. So Π_1 is not in $\mathcal{SET}(monus)$. □

The above results say that $\mathcal{NBL}(unique, eq_b, empty)$ is *conservative* over $\mathcal{NRL}(eq)$ in the sense that it has precisely the same set-theoretic expressive power. On the other hand, $\mathcal{NBL}(unique, eq)$ 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, because of the use of duplicate-free bags to represent sets.

4.2 A Set Language Equivalent to $\mathcal{BQL}$

It was shown in the previous section that $\mathcal{BQL}$ 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. In fact, in order to give the nested relational language the expressive power of $\mathcal{BQL}$, it has to be endowed with natural numbers $\mathbb{N}$ together with multiplication, subtraction, and summation as defined below.

Multiplication $\cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. The semantics of $\cdot$ is multiplication of natural numbers.

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

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

Summation $\sum g : \{s\} \to \mathbb{N}$ where $g : s \to \mathbb{N}$. The semantics is as follows. $\sum g\ \{o_1, \ldots, o_n\} = g(o_1) + \ldots + g(o_n)$. Equivalently, the construct $\Sigma\{\!|e_2 \mid x^s \in e_1|\!\} : \mathbb{N}$ where $e_2 : \mathbb{N}$ and $e_1 : \{s\}$ is also used and is interpreted as $(\sum(\lambda x.e_2))(e_1)$. (We use bag brackets to emphasize the semantics of $\sum$: after f is applied to all elements of a set, all values are counted, even if f produced the same value on a number of elements.)

It is known [8] that adding the booleans and the conditional $cond(e_1, e_2, e_3) :=$ *if* e_1 *then* e_2 *else* e_3 does not add expressiveness to $\mathcal{NRL}(eq)$. We now define a language $\mathcal{NRL}^{\text{nat}}$ as $\mathcal{NRL}(eq)$ with booleans and conditional, enhanced by the type $\mathbb{N}$ of natural numbers and the three operations $\cdot, \dot{-}$ and $\sum$ above.

The rest of this subsection is devoted to proving

Theorem 4.4 $\mathcal{BQL} \simeq \mathcal{NRL}^{\text{nat}}$.

Proof is given in two claims below. First, we need a slightly different kind of conversion between sets and bags. Two additional devices are used: *to_nat* : $\{\!|unit|\!\} \to \mathbb{N}$ takes a bag containing n units to

n and $\mathit{from\_nat} : \mathbb{N} \to \{\!|\mathit{unit}|\!\}$ does the opposite thing. Let s be a complex object type not involving sets. Then $\mathit{to\_set}(s)$ is the type obtained by changing all bag components $\{\!|t|\!\}$ to $\{\mathit{to\_set}(t) \times \mathbb{N}\}$. An object $o : s$ is converted to an object $\mathit{to\_set}_s(o)$ of type $\mathit{to\_set}(s)$. An object $o : \mathit{to\_set}(s)$ is converted to an object $\mathit{from\_set}_s(o)$ of type s. The two conversion functions are defined inductively below.

- $\mathit{to\_set}_{\mathit{unit}} := \mathit{id}$
- $\mathit{to\_set}_{s \times t} := \langle \mathit{to\_set}_s \circ \pi_1, \mathit{to\_set}_t \circ \pi_2 \rangle$
- $\mathit{to\_set}_{\{\!|s|\!\}} := \lambda B.\mathit{bs\_map}(\lambda b.(\mathit{to\_set}_s\ b, \mathit{to\_nat}\{\!|() \mid c \in B, b\ \mathit{eq}\ c|\!\})) \ B$
- $\mathit{from\_set}_{\mathit{unit}} := \mathit{id}$
- $\mathit{from\_set}_{s \times t} := \langle \mathit{from\_set}_s \circ \pi_1, \mathit{from\_set}_t \circ \pi_2 \rangle$
- $\mathit{from\_set}_{\{\!|s|\!\}} := b\_\mu \circ b\_\mathit{map}(b\_\mathit{map}\ \pi_1 \circ b\_\rho_2 \circ \langle \mathit{from\_set}_s \circ \pi_1, \mathit{from\_nat} \circ \pi_2 \rangle)$

Using the above conversion, it can be shown that

Claim 1: $\mathcal{BQL} \subseteq \mathcal{NRL}^{\mathrm{nat}}$.

Proof of Claim 1. Since addition is definable by summation, to prove the proposition, it suffices to show that for each $f : s \to t$ in $\mathcal{BQL}$, there is a $f' : \mathit{to\_set}(s) \to \mathit{to\_set}(t)$ in $\mathcal{NRL}^{\mathrm{nat}}$ such that the diagram below commutes.

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

The right square in the above diagram clearly commutes. Hence we need only to prove that the left square commutes. This is easy by defining f' as follows:

- $!' := \,!$
- $\pi_1' := \pi_1$
- $\pi_2' := \pi_2$
- $Kc' := Kc$
- $K\{\!|\,|\!\}' := K\{\}$
- $\mathit{id}' := \mathit{id}$
- $\langle g, h \rangle' := \langle g', h' \rangle$
- $(g \circ h)' := g' \circ h'$
- $b\_\eta' := s\_\eta \circ \langle \mathit{id}, K1 \circ ! \rangle$
- $b\_\rho_2' := \lambda(x, Y).\{((x, z), n) \mid (z, n) \in Y\}$
- $\mathit{unique}' := s\_\mathit{map}\langle \pi_1, K1 \circ ! \rangle$
- $\mathit{monus}' := \lambda(X, Y).\{(x, m \dot{-} n) \mid (x, m) \in X, (y, n) \in Y, x = y, (m \dot{-} n) \neq 0\}$
- $\uplus' := \cup \circ \langle D, E \rangle$ where $D := \lambda(A, B).\{(a, n + m) \mid (a, n) \in A, (b, m) \in B, a = b\}$ and $E := \lambda(A, B).\{(a, n + m) \mid (a, n) \in B, (b, m) \in A, a = b\}$.
- $b\_\mu' := \lambda A.\{(s, \Sigma\{n \cdot \Sigma\{\mathit{if}\ s = x\ \mathit{then}\ m\ \mathit{else}\ 0 \mid (x, m) \in a\} \mid (a, n) \in A\}) \mid s \in \{x \mid (X, a) \in A, (x, b) \in X\}\}$

- $(b\_map\ g)' := \lambda A.\{(x, \Sigma\{\mathit{if}\ a = x\ \mathit{then}\ n\ \mathit{else}\ 0 \mid (a,n,z) \in B\}) \mid x \in \{x \mid (x,y,z) \in B\}$ where $B := \{(g'\ b, n, b) \mid (b,n) \in A\}$. $\square$

For the inclusion in the other direction, conversions similar to that in Claim 1 are used. Let s be a complex object type not involving sets. Then $to\_bag(s)$ is the type obtained by changing all set brackets to bag brackets and changing $\mathbb{N}$ to $\{\!|unit|\!\}$. Let o be an object of type s. Then it is converted to an object $to\_bag_s(o)$ of type $to\_bag(s)$. Let o be an object of type $to\_bag(s)$, then it is converted to an object $from\_bag_s(o)$ of type s. The conversion functions are defined below.

$$
\begin{array}{ll}
to\_bag_{\mathbb{N}} := from\_nat & from\_bag_{\mathbb{N}} := to\_nat \\
to\_bag_{unit} := id & from\_bag_{unit} := id \\
to\_bag_{s \times t} := \langle to\_bag_s \circ \pi_1, to\_bag_t \circ \pi_2 \rangle & from\_bag_{s \times t} := \langle from\_bag_s \circ \pi_1, from\_bag_t \circ \pi_2 \rangle \\
to\_bag_{\{s\}} := sb\_map(to\_bag_s) & from\_bag_{\{s\}} := bs\_map(from\_bag_s)
\end{array}
$$

Then we have

Claim 2: $\mathcal{NRL}^{\mathrm{nat}} \subseteq \mathcal{BQL}$.

Proof of Claim 2. First note that $\mathcal{NRL}^{\mathrm{nat}} \simeq \mathcal{NRL}(eq, \mathbb{N}, \cdot, \dot{-}, \sum)$, because the booleans and the conditional in $\mathcal{NRL}^{\mathrm{nat}}$ are just devices of convenience and add no power to the language. Hence it suffices to prove that for every $f : s \to t$ in $\mathcal{NRA}(eq_b, not, \mathbb{N}, \cdot, \dot{-}, \sum)$, there is a $f'' : to\_bag(s) \to to\_bag(t)$ in $\mathcal{BQL}$ such that the diagram below commutes.

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

As the right square clearly commutes, we are left to demonstrate that the left square commutes. This can be accomplished by defining f'' as follows, where NAT_n is the bag of exactly n units:

$$
\begin{array}{lll}
!'' := ! & Kn'' := \lambda x.NAT\_n & Kc'' := Kc \\
K\{\}'' := K\{\!|\,|\!\} & \pi_1'' := \pi_1 & \pi_2'' := \pi_2 \\
id'' := id & \langle g, h\rangle'' := \langle g'', h'' \rangle & (g \circ h)'' := g'' \circ h'' \\
s\_\eta'' := b\_\eta & s\_\rho_2'' := b\_\rho_2 & \dot{-}'' := monus \\
\cdot'' := \lambda(X,Y).\{\!|() \mid x \in X, y \in Y|\!\} & (\sum g)'' := b\_\mu \circ b\_map(g'') & (s\_map\ g)'' := unique \circ b\_map(g'') \\
s\_\mu'' := unique \circ b\_\mu & eq'' := eq. & not'' := \lambda R.\{\!|()|\!\}\, monus\ R \\
\cup'' := unique\ \circ \uplus & &
\end{array}
$$

$\square$

4.3 Relational Language with Aggregate Functions, $\mathcal{NRL}^{\mathrm{aggr}}$

The $\sum$ construct introduced to the set language to capture the power of $\mathcal{BQL}$ is often helpful in defining aggregate functions. For example, $\sum(\lambda x.1)$ corresponds to the aggregate COUNT and $\sum\ id$

is TOTAL. However, many aggregate functions are based on rational rather than integer arithmetic. Thus, we suggest that $\mathcal{NRL}^{\mathrm{nat}}$ be extended to a language in which $\mathbb{Q}$, the type of rational numbers, is used as a new base type.

We define the *nested relational language with aggregate functions*, $\mathcal{NRL}^{\mathrm{aggr}}$, as $\mathcal{NRL}(eq)$ enhanced with rational numbers $\mathbb{Q}$; arithmetic operations $+$, $-$, $\cdot$, and $\div$; linear order $\leq_{\mathbb{Q}}$ on rationals; and the summation construct below with a semantics analogous to the summation over natural numbers. For succinctness, we also throw in the booleans and *if-then-else*; but note that they add no expressive power to the language [60].

$$\frac{f : s \to \mathbb{Q}}{\sum f : \{s\} \to \mathbb{Q}}$$

Equivalently, the construct $\Sigma\{\!|e_2 \mid x^s \in e_1|\!\} : \mathbb{Q}$ where $e_2 : \mathbb{Q}$ and $e_1 : \{s\}$ is also used and is interpreted as $(\sum(\lambda x.e_2))(e_1)$.

Many useful aggregate functions can be defined in $\mathcal{NRL}^{\mathrm{aggr}}$. For example:

- "Total the first column of R" is $\mathsf{TOTAL}_1(R) := \sum \{\!|\pi_1\ x \mid x \in R|\!\}$.
- "Average of the first column in R" is $\mathsf{AVG}_1(R) := \mathsf{TOTAL}_1(R) \div \mathsf{COUNT}(R)$.
- "Variance of the first column of R" $\mathsf{VARIANCE}_1(R)$ is

  $$(\sum \{\!|sq(\pi_1\ x) \mid x \in R|\!\} - (sq(\sum \{\!|\pi_1\ x \mid x \in R|\!\}) \div \mathsf{COUNT}(R))) \div \mathsf{COUNT}(R)$$

  where $sq := \lambda y.y \cdot y$.

We consider $\mathcal{NRL}^{\mathrm{aggr}}$ to be a rational reconstruction of SQL for the reason that it is capable of expressing the constructs that contribute to the gap between SQL and relational algebra: nesting, which is needed for GROUP-BY, and aggregate functions. In particular, to justify the claim that SQL cannot express transitive closure, we prove that transitive closure is not definable in $\mathcal{NRL}^{\mathrm{aggr}}$; see Corollary 5.17.

5 Expressive Power of Set Languages, Bag Languages and Aggregate Functions

In this section limitation in the expressive power of $\mathcal{NRL}(eq)$, $\mathcal{NRL}^{\mathrm{aggr}}$, and $\mathcal{BQL}$ is studied. The results of this section are:

- $\mathcal{NRL}^{\mathrm{aggr}}$ and $\mathcal{NRL}^{\mathrm{nat}}$ possess the *conservative extension property.* That is, expressibility of queries in these languages is independent of the height of set nesting in intermediate results.
- The *bounded degree property* is introduced as a tool for investigating inexpressibility. We show that it uniformly applies to a number of recursive queries. We prove that $\mathcal{NRL}(eq)$ has this property. However, it is not known if $\mathcal{NRL}^{\mathrm{aggr}}$ has the bounded degree property.

- By a careful analysis of the normal forms induced by the conservative extension property on $\mathcal{NRL}^{\text{aggr}}$, we show that the same recursive queries are also not expressible in $\mathcal{NRL}^{\text{aggr}}$. As $\mathcal{BQL}$ is clearly a sublanguage of $\mathcal{NRL}^{\text{aggr}}$, these inexpressibility results settle Conjectures 1, 2, and 3.

- We prove that the class of unary arithmetic functions definable in $\mathcal{BQL}$ or $\mathcal{NRL}^{\text{nat}}$ is the class of extended polynomials. As a result, tests for properties which are simultaneously infinite and co-infinite are inexpressible in these languages.

5.1 Conservative Properties of Set and Bag Languages

Let us first explain the idea of *conservative extension.* The set height $ht(s)$ of a type s is defined as the depth of the nesting of set- or bag- brackets in s. The set height $ht(e)$ of an expression e is defined as the maximum of the set heights of all the types that appear in the unique typing derivation of e.

Definition. *A language $\mathcal{L}$ has the* conservative extension property *if any function $f : s \to t$ definable in $\mathcal{L}$ can be expressed in $\mathcal{L}$ using an expression whose set height is at most* $\max(k, ht(s), ht(t))$*, where k is a constant fixed for $\mathcal{L}$.*

In other words, the class of functions computable by a language possessing this property is *independent* of the height of intermediate data structures. Note that if $\mathcal{L}$ has the conservative extension property and k is the fixed constant, then $\mathcal{L}(p)$ also has that property but the fixed constant becomes $\max(k, ht(p))$ for any additional primitive p.

For the rest of this subsection, we use the calculus versions of our languages.

Theorem 5.1 *Let $e : s$ be an expression of $\mathcal{NRC}^{\text{aggr}}$. Then there is an equivalent $\mathcal{NRC}^{\text{aggr}}$ expression $e' : s$ such that $ht(e') \le \max(\{ht(s)\} \cup \{ht(t) \mid t$ is the type of a free variable in $e\})$.*

Proof. We prove conservativity for the language that does not have $\le_{\mathbb{Q}}$ as a primitive. Notice that $\le_{\mathbb{Q}}$ is of height zero; hence adding it does not affect the conservative extension property. We proceed along the lines of Wong [59] by first introducing a strongly normalizing rewrite system and then showing that the normal forms induced by the system do not generate intermediate data of great height. Towards this end, consider the rewrite rules below.

- $(\lambda x.e_1)(e_2) \leadsto e_1[e_2/x]$
- $\pi_i(e_1,\ e_2) \leadsto e_i$
- $\bigcup\{e \mid x \in \{\}\} \leadsto \{\}$
- $\bigcup\{e_1 \mid x \in \{e_2\}\} \leadsto e_1[e_2/x]$
- $\bigcup\{e \mid x \in (e_1 \cup e_2)\} \leadsto \bigcup\{e \mid x \in e_1\} \ \cup\ \bigcup\{e \mid x \in e_2\}$
- $\bigcup\{e_1 \mid x \in \bigcup\{e_2 \mid y \in e_3\}\} \leadsto \bigcup\{\bigcup\{e_1 \mid x \in e_2\} \mid y \in e_3\}$
- $\bigcup\{e \mid x \in (\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3)\} \leadsto \mathit{if}\ e_1\ \mathit{then}\ \bigcup\{e \mid x \in e_2\}\ \mathit{else}\ \bigcup\{e \mid x \in e_3\}$
- $\pi_i\ (\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3) \leadsto \mathit{if}\ e_1\ \mathit{then}\ \pi_i\ e_2\ \mathit{else}\ \pi_i\ e_3$
- $\mathit{if}\ \mathit{true}\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \leadsto e_2$
- $\mathit{if}\ \mathit{false}\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \leadsto e_3$
- $\sum \{\!|e \mid x \in \{\}|\!\} \leadsto 0$
- $\sum \{\!|e \mid x \in \{e'\}|\!\} \leadsto e[e'/x]$

- $\sum \{\!|e \mid x \in \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3|\!\} \leadsto \mathit{if}\ e_1\ \mathit{then}\ \sum \{\!|e \mid x \in e_2|\!\}\ \mathit{else}\ \sum \{\!|e \mid x \in e_3|\!\}$
- $\sum \{\!|e \mid x \in e_1 \cup e_2|\!\} \leadsto \sum \{\!|e \mid x \in e_1|\!\} + \sum \{\!|\mathit{if}\ x \in e_1\ \mathit{then}\ 0\ \mathit{else}\ e \mid x \in e_2|\!\}$
- $\sum \{\!|e \mid x \in \bigcup\{e_1 \mid y \in e_2\}|\!\}$
  $\leadsto \sum \{\!| \sum \{\!|(e \div \sum \{\!| \sum \{\!|\mathit{if}\ x = v\ \mathit{then}\ 1\ \mathit{else}\ 0 \mid v \in e_1|\!\} \mid y \in e_2|\!\}) \mid x \in e_1|\!\} \mid y \in e_2|\!\}$

It is a fact that any equality test $=^s: s \times s \to \mathit{bool}$ can be implemented in terms of equality tests at base types $=^b: b \times b \to \mathit{bool}$, using $\mathcal{NRL}^{\mathrm{aggr}}$ as the ambient language. Moreover, this can be done without using the $\bigcup\{e_1 \mid x \in e_2\}$ construct or the $e_1 \cup e_2$ construct. Furthermore, it can be done without going beyond the height of s. In the rules above, all occurrences of $=$ are to be treated as a sugar for their implementation in terms of equality tests at base types.

Claim. These rewrite rules preserve the meanings of expressions. That is, $e_1 \leadsto e_2$ implies $e_1 = e_2$.

Proof of claim. The proof is straightforward. However, the last rule deserves special attention. Consider the incorrect equation: $\sum \{\!|e \mid x \in \bigcup\{e_1 \mid y \in e_2\}|\!\} = \sum \{\!| \sum \{\!|e \mid x \in e_1|\!\} \mid y \in e_2|\!\}$. Suppose e_2 evaluates to a set of two distinct objects $\{o_1, o_2\}$. Suppose $e_1[o_1/y]$ and $e_1[o_2/y]$ both evaluate to $\{o_3\}$. Suppose $e[o_3/x]$ evaluates to 1. Then the left-hand-side of the "equation" returns 1 but the right-hand-side yields 2. The division operation in the last rule is used to handle duplicates properly.

While the last two rules seem to increase the "character count" of expressions, it should be remarked that $\sum \{\!|e_1 \mid x \in e_2|\!\}$ is always rewritten by these two rules to an expression that decreases in the e_2 position.

Claim. The rewrite system above is strongly normalizing. That is, after a finite number of rewrite steps, we must arrive at an expression to which no rule is applicable.

Proof of claim. The rewrite system above really consists of two orthogonal subsystems. The first contains all the rules except the last two. The second contains the last two rules but not the rest. Thus, to prove the claim, it suffices to provide two non-interfering termination measures for the two subsystems.

For the larger subsystem, we use the following measure. Let φ map 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 $\varphi(x)$. Define $\varphi \leq \varphi'$ if $\varphi(x) \leq \varphi'(x)$ for all x. It is readily seen that $\|\cdot\|\varphi$ is monotonic in φ. Moreover, if $e_1 \leadsto e_2$ via any rule but not the last two, then $\|e_1\|\varphi > \|e_2\|\varphi$. That is, this measure strictly decreases with respect to the first subsystem of rewrite rules.

- $\|x\|\varphi = \varphi(x)$
- $\|\mathit{true}\|\varphi = \|\mathit{false}\|\varphi = \|c\|\varphi = \|()\|\varphi = \|\{\}\|\varphi = 2$
- $\|\pi_1\ e\|\varphi = \|\pi_2\ e\|\varphi = \|\{e\}\|\varphi = 2 \cdot \|e\|\varphi$
- $\|\lambda x.e\|\varphi = \|e\|\varphi[2/x]$
- $\|(\lambda x.e)(e')\|\varphi = \|e\|\varphi[\|e'\|\varphi/x] \cdot \|e'\|\varphi$
- $\|e_1\ \cup\ e_2\|\varphi = \|(e_1, e_2)\|\varphi = \|e_1 + e_2\|\varphi = \|e_1 - e_2\|\varphi = \|e_1 \cdot e_2\|\varphi = \|e_1 \div e_2\|\varphi = \|e_1 =_b e_2\|\varphi = 1 + \|e_1\|\varphi + \|e_2\|\varphi$

- $\| \bigcup\{e' \mid x \in e\} \| \varphi = (\|e'\|\varphi[\|e\|\varphi/x] + 1) \cdot \|e\|\varphi$
- $\| \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \| \varphi = \|e_1\|\varphi \cdot (1 + \|e_2\|\varphi + \|e_3\|\varphi)$
- $\| \sum \{\!|e' \mid x \in e|\!\} \| \varphi = (\|e'\|\varphi[\|e\|\varphi/x] + 1) \cdot \|e\|\varphi$.

For example, the left-hand-side of the rule $\bigcup\{e_1 \mid x \in \bigcup\{e_2 \mid y \in e_3\}\} \leadsto \bigcup\{\bigcup\{e_1 \mid x \in e_2\} \mid y \in e_3\}$ has measure $\|e_1\|\varphi \cdot \|e_2\|\varphi \cdot \|e_3\|\varphi + \|e_1\|\varphi \cdot \|e_3\|\varphi + \|e_2\|\varphi \cdot \|e_3\|\varphi + \|e_3\|\varphi$. However, the right-hand-side has measure $\|e_1\|\varphi \cdot \|e_2\|\varphi \cdot \|e_3\|\varphi + \|e_3\|\varphi$. The latter is clearly smaller.

For the second subsystem, we need a more complex measure, which we define in two steps. For the first step, let θ be a function that maps variables to natural numbers greater than 1. Let $\theta[n/x]$ be the function that maps x to n and agrees with θ on other variables. Let $\|e\|\theta$ be defined as below. Then $\|e\|\theta$ is monotone in θ. Moreover, if $e_1 \leadsto e_2$ via any rules, then $\|e_1\|\theta \geq \|e_2\|\theta$.

- $\|\mathit{true}\|\theta = \|\mathit{false}\|\theta = \|c\|\theta = \|()\|\theta = \|\{\}\|\theta = 2$
- $\|\pi_1\ e\|\theta = \|\pi_2\ e\|\theta = \|e\|\theta$
- $\|x\|\theta = \theta(x)$
- $\|\lambda x.e\|\theta = \|e\|\theta[2/x]$
- $\|(\lambda x.e_1)(e_2)\|\theta = \max(\|e_2\|\theta, \|e_1\|\theta[\|e_2\|\theta/x])$
- $\|\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3\|\theta = \max(\|e_1\|\theta, \|e_2\|\theta, \|e_3\|\theta)$
- $\|\{e\}\|\theta = 1 + \|e\|\theta$
- $\|e_1 \cup e_2\|\theta = 1 + \max(\|e_1\|\theta, \|e_2\|\theta)$
- $\|\bigcup\{e_1 \mid x \in e_2\}\|\theta = (\|e_1\|\theta[\|e_2\|\theta/x])^{\|e_2\|\theta}$
- $\|(e_1, e_2)\|\theta = \|e_1 + e_2\|\theta = \|e_1 - e_2\|\theta = \|e_1 \cdot e_2\|\theta = \|e_1 \div e_2\|\theta = \|e_1 =_b e_2\|\theta = \max(\|e_1\|\theta, \|e_2\|\theta)$
- $\|\sum \{\!|e_1 \mid x \in e_2|\!\}\|\theta = \max(\|e_2\|\theta, \|e_1\|\theta[\|e_2\|\theta/x])$

For the second step, let σ denote an infinite tuple $(\ldots, \sigma(1), \sigma(0))$ of natural numbers with finitely many non-zero components. These tuples are ordered left-to-right lexicographically. Furthermore, since each tuple has only finitely many non-zero components, the ordering is well founded. Let $\sigma_1 * \sigma_2$ denote the tuple σ obtained by component-wise summation of σ_1 and σ_2. Let $\sigma[n]$ denote the tuple σ' such that $\sigma'(n) = \sigma(n) + 1$ and $\sigma'(m) = \sigma(m)$ for $m \neq n$. Let ϕ be a function mapping variables to tuples σ's. Let $\phi[\sigma/x]$ map x to the tuple σ and agree with ϕ on other variables. Let $\|e\|\phi\theta$ be defined as below. Then $\|e\|\phi\theta$ is monotone in both ϕ and θ. Furthermore, if $e_1 \leadsto e_2$, then $\|e_1\|\phi\theta \geq \|e_2\|\phi\theta$. More importantly, if $e_1 \leadsto e_2$ via the last two rewrite rules above, then $\|e_1\|\phi\theta > \|e_2\|\phi\theta$. (For example, the measure for the left-hand-side of the last and most complicated rule simplifies to $(\|e\|\phi\theta * \|e_1\|\phi\theta * \|e_2\|\phi\theta)[\|e_1\|\theta^{\|e_2\|\theta}]$. On the other hand, the measure for the right-hand-side simplifies to $(\|e\|\phi\theta * \|e_1\|\phi\theta * \|e_2\|\phi\theta)[\|e_1\|\theta][\|e_1\|\theta][\|e_2\|\theta][\|e_2\|\theta]$. The latter is clearly smaller than the former.) Thus this measure strictly decreases for the last two rules. That is, it strictly decreases for the second subsystem of rules and decreases (but not necessarily strictly) for the first subsystem of rules.

- $\|x\|\phi\theta = \phi(x)$
- $\|\lambda x.e\|\phi\theta = \|e\|\phi[(\ldots,0)/x]\theta[2/x]$
- $\|(\lambda x.e_1)(e_2)\|\phi\theta = \|\bigcup\{e_1 \mid x \in e_2\}\|\phi\theta = \|e_2\|\phi\theta * \|e_1\|(\phi[\|e_2\|\phi\theta/x])(\theta[\|e_2\|\theta/x])$
- $\|\mathit{true}\|\phi\theta = \|\mathit{false}\|\phi\theta = \|c\|\phi\theta = \|()\|\phi\theta = \|\{\}\|\phi\theta = (\ldots,0)$
- $\|\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3\|\phi\theta = \|e_1\|\phi\theta * \|e_2\|\phi\theta * \|e_3\|\phi\theta$
- $\|\pi_1\ e\|\phi\theta = \|\pi_2\ e\|\phi\theta = \|\{e\}\|\phi\theta = \|e\|\phi\theta = \|e\|\phi\theta$
- $\|e_1 \cup e_2\|\phi\theta = \|(e_1,e_2)\|\phi\theta = \|e_1 =_b e_2\|\phi\theta = \|e_1 + e_2\|\phi\theta = \|e_1 - e_2\|\phi\theta = \|e_1 \cdot e_2\|\phi\theta = \|e_1 \div e_2\|\phi\theta = \|e_1\|\phi\theta * \|e_2\|\phi\theta$
- $\|\sum \{\!|e_1 \mid x \in e_2|\!\}\|\phi\theta = (\|e_2\|\phi\theta * \|e_1\|\phi[\|e_2\|\phi\theta/x]\theta[\|e_2\|\theta/x])[\|e_2\|\theta]$

The termination measure for the entire rewrite system above can now be defined as $\|e\|\varphi\phi\theta = (\|e\|\phi\theta, \|e\|\varphi)$, ordered left-to-right lexicographically. Then $\|e\|\varphi\phi\theta$ is monotone in all of φ, ϕ, and θ. Furthermore, if $e_1 \leadsto e_2$, then $\|e_1\|\varphi\phi\theta > \|e_2\|\varphi\phi\theta$. Since orderings for both components are well-founded, so is the lexicographic ordering. Therefore, the rewrite system above is strongly normalizing.

Next we analyze the normal forms induced by the rules above.

Claim. Let $e : s$ be an expression of $\mathcal{NRL}^{\text{aggr}}$ in normal form. Then $ht(e) \leq \max(\{ht(s)\} \cup \{ht(t) \mid t$ is the type of a free variable occurring in $e\})$.

Proof of claim. The proof is a routine induction on the structure of normal forms. Since the rewrite system always leads to some normal forms, this completes the proof of the theorem. □

One of our goals is to demonstrate that division, enumeration of natural numbers from 0 to n for a given n, etc. are not expressible in $\mathcal{BQL}$. Observe that $\div$ is critical for achieving conservative extension in the nested relational language endowed with rationals. Therefore, it is possible that the set language equivalent to $\mathcal{BQL}$, which lacks division, may not possess the conservative extension property. Fortunately, this is not the case because $\mathcal{NRL}^{\text{nat}}$ has sufficient horsepower to compute a linear order at all types, as shown by us in another paper [37].

Proposition 5.2 *For each type s it is possible to define a function $\leq_s$: $s \times s \to \{unit\}$ in $\mathcal{NRC}^{\text{nat}}$ which defines a linear order on the elements of s. Moreover, it can be defined in such a way that $ht(\leq_s) = ht(s)$.* □

In the next theorem, this linear order is exploited to show that functions in $\mathcal{NRC}^{\text{nat}}$, and hence $\mathcal{BQL}$, do not depend on intermediate data structures.

Theorem 5.3 *Let $e : s$ be an expression of $\mathcal{NRC}^{\text{nat}}$. Then there is an equivalent expression $e' : s$ such that $ht(e') \leq \max(\{ht(s)\} \cup \{ht(t) \mid t$ is the type of a free variable in $e\})$.*

Proof. It suffices to replace the last rule used in Theorem 5.1 by the following: $\sum \{\!|e \mid x \in \bigcup\{e_1 \mid y \in e_2\}|\!\} \leadsto \sum \{\!| \sum \{\!| \mathit{if}\ (\sum \{\!| \mathit{if}\ x \in e_1[w/y]\ \mathit{then}\ (\mathit{if}\ w = y\ \mathit{then}\ 0\ \mathit{else}\ (\mathit{if}\ w \leq y\ \mathit{then}\ 1\ \mathit{else}\ 0))\ \mathit{else}\ 0 \mid w \in e_2|\!\}) = 0\ \mathit{then}\ e\ \mathit{else}\ 0 \mid x \in e_1|\!\} \mid y \in e_2|\!\}$.

The problem of $\sum \{\!|e \mid x \in \bigcup\{e_1 \mid y \in e_2\}|\!\}$ is that the e_1 generated from different y's in e_2 may have nonempty intersection. In the last rule of Theorem 5.1, the duplicates are dealt with by dividing their "contribution" to the final sum by the number of duplicates. The idea of the new rule above is different. If y_1 and y_2 produce e_1's with nonempty intersection, the overlapping values are counted only once. This is achieved by using the linear order of Proposition 5.2 to count those from the smaller of y_1 and y_2. □

The conservative extension property was first studied by Paredaens and Van Gucht [46] and later by Van den Bussche [55]. They proved that $\mathcal{NRC}(eq)$ has it when the input and output are restricted to flat relations. It was then extended by Wong [59] to any input and output. More recently, Suciu [50] managed to prove the remarkable theorem that $\mathcal{NRC}(eq, \mathit{bfix})$, note the absence of natural numbers, has the conservative extension property when input and output are restricted to flat relations. Here *bfix* is a bounded version of the fixpoint operator. When added to first order logic, it yields a language equivalent to datalog with negation.

The results presented in this section show that, with very little extra, conservative extension property holds at any input/output in the presence of aggregate functions, transitive closure, and bounded fixpoint. This is a very significant improvement of these previous results.

Grumbach and Milo [21] obtained a non-collapsing hierarchy theorem. Let $\mathit{gen} : \mathbb{N} \to \{\mathbb{N}\}$ be a primitive which takes the number n to the set $\{0, \ldots, n\}$. Their theorem is equivalent to saying that for any k and i, there is an expression e in $\mathcal{NRC}^{\mathrm{nat}}(\mathit{gen}, \mathit{powerset})$ where $ht(e)$ is at most k and the *number* of *powerset* operators along any path in e is at most $i + 2$ such that there is no equivalent e' of height at most k and the number of *powerset* operators along any path in e' is at most i. This is a result on a different dimension of conservativity. It is a complement, rather than a contradiction, of the last part of the corollary above.

The need for *gen* in the preceding discussion may not be obvious. The language of Grumbach and Milo is a language for bags, in which the natural numbers are simulated as bags of *unit*. Thus they could use the *powerset* operation in their language to enumerate small bags of *unit* (the small numbers) given a big bag of *unit* (a big number). In contrast, $\mathcal{NRC}^{\mathrm{nat}}(\mathit{powerset})$ is a set language. Thus its *powerset* operator cannot be used to generate small numbers from a big number. As we shall see later, the *gen* operator fills this gap nicely.

By the conservative extension property, the class of functions on flat relations computed by $\mathcal{NRL}^{\mathrm{aggr}}$ is precisely that computed by flat relational algebra endowed with the same primitives. This has a practical significance because it implies that $\mathcal{NRL}^{\mathrm{aggr}}$ can be used as a convenient interface to databases that speak SQL. A theoretically more interesting consequence is that every function of type $\{\!|\mathit{unit}|\!\} \to \{\!|\mathit{unit}|\!\}$ in $\mathcal{BQL}$ corresponds to very simple arithmetic. This fact is exploited in the next section, where arithmetic properties of bag query languages are studied.

5.2 Arithmetic Power of $\mathcal{BQL}$

As seen earlier, natural numbers are present in $\mathcal{BQL}$ as objects of type $\{\!|\mathit{unit}|\!\}$. So it is possible to translate $\mathcal{BQL}$ into a *set* language augmented by either rational or natural numbers and some arithmetic. In this section the conservative extension results from the previous section are used to investigate the arithmetic power of $\mathcal{BQL}$ and $\mathcal{NRL}^{\mathrm{aggr}}$. We show that no property of natural numbers that is simultaneously infinite and co-infinite can be tested in either language. This result is particularly surprising for the language augmented by rational numbers and *division*, since it implies the inexpressibility of parity test even when division by two is expressible. Next we show that if $\dot{-}$ is removed from the list of primitives of the language augmented by rationals, then there is no expression that defines the usual ordering on rationals. Finally, we give a complete characterization of unary arithmetic functions in $\mathcal{BQL}$.

Proposition 5.4 *Let $\mathcal{U}$ be a property of natural numbers, that is both infinite and co-infinite. That is, $\mathcal{U} \subseteq \mathbb{N}$ and both $\mathcal{U}$ and $\mathbb{N} - \mathcal{U}$ are infinite. Then the membership test for $\mathcal{U}$ cannot be expressed in $\mathcal{NRL}(\mathbb{Q}^+, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, eq)$, where $\mathbb{Q}^+$ is the type of non-negative rationals.*

Proof. Suppose there is an expression $e : \mathbb{Q}^+ \to \mathbb{Q}^+$ that tests for membership in $\mathcal{U}$. That is, if $n \in \mathcal{U}$, then $e(n) = 1$ and if $n \in \mathbb{N} - \mathcal{U}$, then $e(n) = 0$. (We are not interested in what e returns on elements of $\mathbb{Q}^+ - \mathbb{N}$.) We may assume without loss of generality that e is defined everywhere. That is, division by zero cannot occur in the course of the evaluation of e.

An expression e of type $\mathbb{Q}^+ \to \mathbb{Q}^+$ is called a plus-expression (zero-expression) if there is a number n, depending on e, such that for every $x \geq n$, it is the case that $e(x) > 0$ ($e(x) = 0$). It is enough to prove that any expression e of type $\mathbb{Q}^+ \to \mathbb{Q}^+$ that is defined everywhere is either a plus- or a zero-expression, because testing membership in $\mathcal{U}$ cannot be such. In fact, we show that for any plus-expression, there are two polynomial functions $p(x)$ and $q(x)$ with rational coefficients such that for any $x \geq n$, it is the case that $e(x) = p(x)/q(x)$ and $p(x), q(x) \geq 0$.

Let e be of type $\mathbb{Q}^+ \to \mathbb{Q}^+$. Since $\dot{-}$ can be expressed if a linear order on $\mathbb{Q}^+$ is present, and such a linear order as a primitive has height 0, the language $\mathcal{NRL}(\mathbb{Q}^+, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, eq)$ has the conservative extension property. Hence, e can be considered to be a height-zero expression. That is, it is obtained from its only free variable and constants by operations $+$, $\dot{-}$, $\cdot$, and $\div$. Observe that there is a simple way to code conditionals. Every condition can be reduced to $e' = e''$. For the equality test, $e' = e''$, observe that $(1 \dot{-} (e' \dot{-} e'')) \cdot (1 \dot{-} (e'' \dot{-} e'))$ returns 1 if $e' = e''$ and 0 otherwise. Therefore, we may assume that in any if-then-else statement the condition can be either 1 or 0. But then *if c then f_1 else f_2* is equivalent to $c \cdot f_1 + (1 \dot{-} c) \cdot f_2$. This shows that conditionals can be removed from any expression of type $\mathbb{Q}^+ \to \mathbb{Q}^+$.

We proceed to prove the main claim by induction on the structure of e. The base case and the cases $e = e_1 + e_2$, $e = e_1 \cdot e_2$ and $e = e_1 \div e_2$ are straightforward. Let $e = e_1 \dot{-} e_2$. The only case that is not immediate is when both e_1 and e_2 are plus-expressions given by $p_1(x)/q_1(x)$ and $p_2(x)/q_2(x)$ for x greater than n_1 and n_2 respectively. Consider $f(x) = p_1(x) \cdot q_2(x) - p_2(x) \cdot q_1(x)$. If f is the constant function 0, then e is a zero-expression. Otherwise, let x_f be the maximal root of the polynomial f. There are two cases. If $f(y) > 0$ whenever $y > x_f$, then for every $x \geq \max\{n_1, n_2, x_f\} + 1$, we have $p_1(x)/q_1(x) - p_2(x)/q_2(x) > 0$ and therefore $e(x)$ is a plus-expression given by $(p_1(x) \cdot q_2(x) - p_2(x) \cdot q_1(x))/(q_1(x) \cdot q_2(x))$. If $f(y) < 0$ whenever $y > x_f$, then for every $x \geq \max\{n_1, n_2, x_f\} + 1$, we have $p_1(x)/q_1(x) - p_2(x)/q_2(x) < 0$ and so $e(x) = 0$, a zero-expression. The claim, and the proposition are thus proved. □

It is well known that the relational algebra cannot express parity test [12]. By the results of Paredaens and Van Gucht [46] and Wong [59], 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 $\mathcal{NRL}^{\text{nat}}$, which is a sublanguage of $\mathcal{NRL}(\mathbb{Q}^+, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, eq)$, and hence not expressible in $\mathcal{BQL}$. This is another consequence of the conservative extension property.

Corollary 5.5 *Parity test on numbers is not expressible in* $\mathcal{NRL}^{\mathrm{nat}}$. $\square$

This settles the variant of Conjecture 1 for parity test on numbers. In a more limited setting, where there are no nested bags, it was also proved by Grumbach and Milo [21]. We used conservative extension to obtain the more general result above.

The corollary above says that it is impossible to test whether a natural number is even or odd. However, it is possible to test whether a set has an even or odd number of elements by exploiting the linear order: $odd(R) := \bigcup\{if\ \sum\{if\ x < y\ then\ 1\ else\ 0 \mid y \in R\} = \sum\{if\ x > y\ then\ 1\ else\ 0 \mid y \in R\}\ then\ \{()\}\ else\ \{\} \mid x \in R\}\ = \{()\}$. As a consequence, $\mathcal{BQL}$ cannot test whether a bag contains an even or odd number of elements, but it can test whether a bag contains an even or odd number of *distinct* elements. Using the same technique we can split a set into k equal parts, even though division by k is undefinable. However, this is based on the assumption that a linear order is given for all base types. In the absence of linear order, parity of cardinality is no longer definable; see Corollary 5.16.

As another application of the conservative extension property, we show that in the absence of $\dot{-}$, the usual order on rational numbers is no longer expressible.

Proposition 5.6 *The language obtained from* $\mathcal{NRL}^{\mathrm{aggr}}$ *by removing the* $\leq_{\mathbb{Q}}$ *primitive cannot express the ordering* $\leq_{\mathbb{Q}}$ *on* $\mathbb{Q}$.

Proof. We claim the following. For any expression $g : \mathbb{Q} \to \mathbb{Q}$ defined by using $+$, $\cdot$, $\div$, $=$, if-then-else, constants and *minus*, there exist two polynomials $p(x)$ and $q(x)$ with rational coefficients such that $g(x)$ coincides with $p(x)/q(x)$ almost everywhere; that is, $g(x) \neq p(x)/q(x)$ for only finitely many $x \in \mathbb{Q}$. To prove this claim, we proceed by induction on the structure of an expression g. The base case is immediate. The induction step easily goes through the arithmetic operations. Let $g := if\ c\ then\ g_1\ else\ g_2$. The condition c is $e' = e''$. By induction hypothesis, $e' = p'/q'$, $e'' = p''/q''$, $g_1 = p_1/q_1$, and $g_2 = p_2/q_2$ almost everywhere. Notice that c is either true almost everywhere or false almost everywhere. Indeed, consider $r := p' \cdot q'' - p'' \cdot q'$. If r is the constant function 0, then c may be false only in some of the points in which e' and e'' do not coincide with their polynomial representations. If r is not the constant function 0, then r has finitely many roots and therefore c is true only in finitely many points. Thus g coincides with either p_1/q_1 or p_2/q_2 almost everywhere.

Now, if $\leq_{\mathbb{Q}}$ is definable, then so is the following function g from $\mathbb{Q}$ to $\mathbb{Q}$: $g(x) = 0$ if $x \leq 1$ and $g(x) = 1$ if $x > 1$. It follows from the claim above and conservative extension that g must coincide with ratios of polynomials almost everywhere. However, this is not the case since g has infinitely many roots but is not zero almost everywhere. This contradiction shows that $\leq_{\mathbb{Q}}$ is not expressible. $\square$

Combining techniques of Propositions 5.6 and 5.4 we can show the following.

Corollary 5.7 *Properties of natural numbers that are simultaneously infinite and coinfinite cannot be tested in* $\mathcal{NRL}^{\mathrm{aggr}}$. $\square$

We now turn to the nested relational language $\mathcal{NRL}^{\mathrm{nat}}$, which is equivalent to $\mathcal{BQL}$. Using the conservative extension property, we can prove the following result in a straighforward manner.

Proposition 5.8 *The functions from $\mathbb{N}$ to $\mathbb{N}$ that coincide with polynomials in all but finitely many points are exactly the unary arithmetic functions expressible in $\mathcal{BQL}$.* $\square$

Corollary 5.9 *None of the following functions is expressible in $\mathcal{BQL}$: parity test, division by a constant, bounded summation, bounded product, and $f : \mathbb{N} \to \{\!|\mathbb{N}|\!\}$ such that $f(n) = \{\!|0, 1, \ldots, n|\!\}$.*

Proof. That parity test is not expressible follows either from Proposition 5.4 or the previous proposition. Suppose integer division-by-two, $div_2(n) = \lfloor n/2 \rfloor$, is definable. Then n is even iff $n = 2 \cdot div_2(n)$, which shows inexpressibility of div_2. If a bounded summation is definable, then $f(n) = \sum_{i=0}^{n}$ *if* $2 \cdot i = n$ *then* 1 *else* 0 is a parity test. Similarly, if bounded product is definable, then $f(n) = \prod_{i=0}^{n}$ *if* $2 \cdot i = n$ *then* 2 *else* 1 gives us a parity test. Finally, since all operations in $\mathcal{NRL}^{\rm nat}$ are polynomial, the size of the output of any function $f : \mathbb{N} \to t$ is bounded by a constant, because the size of the input is 1, which proves the inexpressibility of the generator of smaller numbers. $\square$

Therefore, the arithmetic of $\mathcal{BQL}$ is quite limited. In Section 6, where non-polynomial primitives are studied, we show that two extended languages give rise to all elementary and primitive recursive functions respectively.

5.3 Recursive Queries and Bounded Degree Property

In this subsection, we first define two sample queries and show that they are at most as hard as deterministic transitive closure in a language having at least the power of the flat relational algebra or first-order logic. Then we define the bounded degree property and show that it implies a number of inexpressibility results in a uniform fashion. Finally we prove that this property holds in $\mathcal{NRL}$.

Definition.

- *chain* : $\{s \times s\} \to$ *bool is a query that takes a graph and returns true iff the graph is a chain. That is, it returns true iff the graph is a tree such that the out-degree of each node is at most 1.*
- *bbtree* : $\{s \times s\} \to$ *bool is a query that takes a graph and returns true iff the graph is a balanced binary tree. That is, it returns true iff the graph is a binary tree in which all paths from the root to the leaves have the same length.*
- *dtc* : $\{s \times s\} \to \{s \times s\}$ *is the deterministic transitive closure. That is, if $G = \langle V, E\rangle$ is a digraph, then $dtc(G) = \langle V, E'\rangle$ where $(v_1, v_k) \in E'$ iff there is a path $(v_1, v_2) \in E, \ldots, (v_{k-1}, v_k) \in E$ such that v_{i+1} is a unique descendant of v_i, $i = 1, \ldots, k-1$.* See Immerman [24].

We first prove that the first two sample queries are no harder than the third.

Proposition 5.10 *Let $\mathcal{L}$ be a language that has at least the power of the relational algebra. Then chain and bbtree are expressible in $\mathcal{L}(dtc)$.*

Proof. The result for *chain* is straightforward. It also follows from a recent result of Etessami that *chain* is first-order complete for DLOGSPACE [16]. In particular, $\mathcal{NRL}(eq, dtc) \simeq \mathcal{NRL}(eq, chain)$. Now we sketch the proof of the expressibility of *bbtree*. Given a graph $G = \langle V, E\rangle$, let $G^* = \langle V, E^*\rangle$

be $dtc(G^{-1})$, where $G^{-1} = \langle V, \{(v,u) \mid (u,v) \in E\}\rangle$. Define root as a node with in-degree zero and leaves as nodes with out-degree zero in G. Then it is not hard to see that G is a binary tree iff the following is true. In G all nodes which are not leaves or roots have in-degree one and out-degree two; there is exactly one root that has out-degree two and all leaves have in-degree one. In G^*, there are no loops and for every leaf l and the root r, $(l,r) \in E^*$.

To check that G is balanced, one should verify that all maximal paths from leaves to the root have the same length. To do this, for any leaf l consider the set $P(l) = \{l' \mid l' = l \ \vee \ l' = r \ \vee \ ((l,l') \in E^* \ \wedge \ (l',r) \in E^*)\}$. To check if $card(P(l_1)) = card(P(l_2))$, define a binary relation R on $P(l_1) \times P(l_2)$ by $(v_1,v_2)R(v_1',v_2') \Leftrightarrow (v_1',v_1) \in E \ \wedge \ (v_2',v_2) \in E$. Then $card(P(l_1)) = card(P(l_2))$ iff $((l_1,l_2),(r,r)) \in dtc(R)$. □

It follows that

Corollary 5.11 *Let $\mathcal{L}$ be a language that has at least the power of the relational algebra. If chain is not expressible in $\mathcal{L}$, then none of the following are expressible in $\mathcal{L}$: dtc, transitive closure, tests for connectivity of directed and undirected graphs, testing whether a graph is a tree, testing for acyclicity.* □

Let $G = \langle V, E\rangle$ be a graph. Define $\textit{in-deg}(v) = card(\{v' \mid (v',v) \in E\})$ and $\textit{out-deg}(v) = card(\{v' \mid (v,v') \in E\})$. The *degree set* of G, $deg(G)$, is defined as $\{\textit{in-deg}(v) \mid v \in V\} \cup \{\textit{out-deg}(v) \mid v \in V\} \subseteq \mathbb{N}$. One of the reasons why most recursive queries are not first-order definable is that they may take in a graph[1] whose degree set contains only small integers and may return a graph whose degree set is large. The definition below captures this intuition.

Definition. *Let q be a graph query. It is said to have the* bounded degree property *if for any number k there exists a number $c(q,k)$, depending on q and k only, such that the cardinality of the degree set of $q(G)$, $card(deg(q(G)))$, is at most $c(q,k)$ for any graph G satisfying $deg(G) \subseteq \{0,1,\ldots,k\}$. A language $\mathcal{L}$ is said to have the bounded degree property at type s if any $\mathcal{L}$-definable graph query $q : \{s \times s\} \to \{s \times s\}$ has it.*

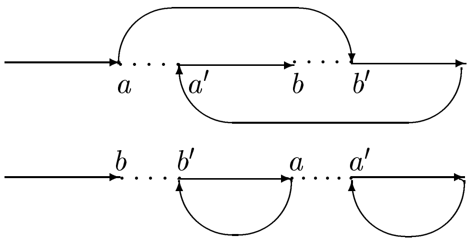
The bounded degree property can be used to prove various inexpressibility results in a uniform fashion. It is also easier to apply the bounded degree property than tools such as games or Hanf's lemma.

Theorem 5.12 *Let $\mathcal{L}$ be a language that has at least the power of the relational algebra. Suppose $\mathcal{L}$ has the bounded degree property at type s. Then neither chain : $\{s \times s\} \to bool$ nor bbtree : $\{s \times s\} \to bool$ is expressible in $\mathcal{L}$.*

Proof. We offer a proof by picture. The details and transformations are readily expressed by first-order formulas.

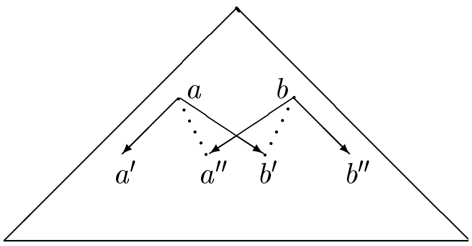
Assume that *chain* is definable. Then it is possible to define an expression that, when given a chain as an input, returns its transitive closure. As shown below, using chain it is possible to determine if a precedes b by re-arranging two edges and checking if the resulting graph is a chain. First, edges from a and b to their successors a' and b' are removed and then two edges are added: one from a to b' and the other from the node with no outgoing edges to a'.

[1] We use graphs for the simplicity of exposition. Relational structures of arbitrary finite arity can be used.



But this contradicts the bounded degree property as we started with an $n-$node graph whose degree set is $\{0, 1\}$ and ended up with $\{0, 1, \ldots, n\}$.

If *bbtree* is definable, it is possible to determine if two nodes in a balanced binary tree are at the same level by re-arranging two edges as shown below and checking if the result is still a balanced binary tree.



Again, we start with an $n-$node graph whose degree set is $\{0, 1, 2\}$ and, making cliques of the nodes at the same level, end up with a graph whose degree set has cardinality $\log_2(n + 1)$. □

Having seen the power of the bounded degree property, we now prove that first-order logic has it. By conservativity, it means that it holds in the nested relational language $\mathcal{NRL}(eq)$.

Theorem 5.13 *Any first-order definable graph query has the bounded degree property.*

To be more specific, we view graph queries as relational calculus expressions of the form $\{(a, b) \mid F(a, b)\}$ where F is a first-order formula in the language that contains a binary predicate symbol E and equality. To evaluate such a query Q on a graph $G = \langle V, E \rangle$, one can assume that all quantified variables in F range over V, cf. [23]. We can also view them as formulae $\forall a \forall b. E'(a, b) \leftrightarrow F(a, b)$ where E' is the predicate for the output graph, and $F(\cdot, \cdot)$ is as above. Theorem 5.13 says that such definable queries have the bounded degree property.

Proof of Theorem 5.13. Let Q be a graph query given by the first-order formula $F(\cdot, \cdot)$ in the language that contains E and equality. That is, $(a, b) \in E'$ iff $F(a, b)$, where E' is the set of edges of the output graph. By a neighborhood of radius r of x in E we mean the set of all nodes whose distance from x (that is, the length of a minimal path in the symmetric closure of E) does not exceed r. We denote the r-neighborhood of x by $N_r(x)$. By $N_r(X)$ we mean $\bigcup_{x \in X} N_r(x)$. According to Gaifman [19], F is equivalent to a Boolean combination of formulae with a and b as free variables in which all quantifiers are bounded to some neighborhoods of a and b. Moreover, the maximal radius of those neighborhoods, r, is determined by F.

If $deg(G) \subseteq \{0, \ldots, k\}$, then it is possible to find the number q_r of all non-isomorphic neighborhoods around a node of radius up to r. In fact, $q_r \leq p_r 2^{p_r^2}$ where $p_r = (2k+1)^r$ is an upper bound on the size of $N_r(x)$.

Define an equivalence relation $\approx$ on the nodes by letting $a \approx b$ iff $N_{2r+1}(a)$ and $N_{2r+1}(b)$ are isomorphic. Note that if $a \approx b$, then $N_d(a)$ and $N_d(b)$ are isomorphic for any $d \leq 2r+1$. Now consider the partition $X_1, \ldots, X_s$ of the set of nodes into $\approx$-equivalence classes. Since $deg(G) \subseteq \{0, \ldots, k\}$, we obtain $s \leq q_{2r+1}$.

Let a_1, a_2 belong to the same class X_i. If $b \notin N_{2r+1}(a_1) \cup N_{2r+1}(a_2)$, then $N_r(a_1, b)$ is the disjoint union of $N_r(a_1)$ and $N_r(b)$ and $N_r(a_2, b)$ is the disjoint union of $N_r(a_2)$ and $N_r(b)$. Hence, $N_r(a_1, b)$ and $N_r(a_2, b)$ are isomorphic. In particular, $(a_1, b) \in E'$ iff $(a_2, b) \in E'$. In F all quantified variables are bounded to the neighborhoods of its free variables of radius at most r. Since these neighborhoods are isomorphic when free variables are a_1, b and a_2, b, and since evaluating first-order formulae on isomorohic models gives the same result, the statement follows.

Now let $Y_a = \{b \mid (a,b) \in E'\}$. Then there exists a constant d_i that depends only on r and k such that $\mid card(Y_{a_1}) - card(Y_{a_2}) \mid \leq d_i$ whenever $a_1,\ a_2 \in X_i$. Indeed, for elements b outside of $N_{2r+1}(a_1) \cup N_{2r+1}(a_2)$, (a_1, b) iff (a_2, b), and hence the only difference is in the edges either inside or between those neighborhoods. The maximal difference therefore is bounded by the doubled size of such a neighborhood, that is, by $2(2k+1)^{2r+1}$.

This shows that the number of different outdegrees for nodes that belong to the same $\approx$-equivalence class X_i is bounded by a constant that depends only on r and k. Since the number of $\approx$-equivalence classes is also bounded by the constant q_{2r+1} depending on k and r only, we obtain that the number of possible different outdegrees is at most the product of these constants, and hence determined by k and r. Since r depends only on Q (it can be calculated by a procedure suggested in [19]), the number of distinct outdegrees in E' is bounded by a constant that depends only on k and f. The proof for indegrees is similar. □

Corollary 5.14
- *The relational algebra queries have the bounded degree property.*
- *$\mathcal{NRL}(eq)$ has the bounded degree property at base types.*
- *chain, bbtree, and the other queries listed in Corollary 5.11 are not expressible in $\mathcal{NRL}$.* □

5.4 Expressiveness of $\mathcal{BQL}$ and $\mathcal{NRL}^{\text{aggr}}$

If we could prove that $\mathcal{NRL}^{\text{aggr}}$ possesses the bounded degree property, we would have shown that every query listed in Corollary 5.11 is neither $\mathcal{NRL}^{\text{aggr}}$- nor $\mathcal{BQL}$-definable. Unfortunately, it is still an open problem as to whether $\mathcal{NRL}^{\text{aggr}}$ has the bounded degree property. To the best of our knowledge, there is no known logic capturing the language $\mathcal{NRL}^{\text{aggr}}$, not even its flat fragment. The proof of the bounded degree property for $\mathcal{NRC}$ is based on Gaifman's result about local formulae [19]. That result was proved by quantifier elimination. This poses a problem if we try to prove the bounded degree property for flat types in $\mathcal{NRL}^{\text{aggr}}$.

Inexpressibility of recursive queries in languages with aggregates was studied by Consens and Mendelzon [15]. They showed that transitive closure is not expressible in a first-order language with aggregate functions, provided DLOGSPACE is strictly included in NLOGSPACE.

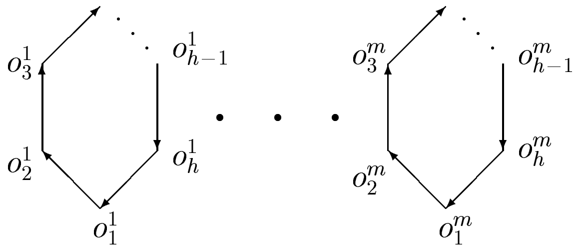


Figure 4: A multi-cycle

However, there is no simple proof of the main conjectures based on this kind of complexity arguments. For example, Conjecture 1 states that a DLOGSPACE-complexity query is not expressible in $\mathcal{NRL}^{\text{aggr}}$. If it could be shown that the complexity of $\mathcal{NRL}^{\text{aggr}}$ queries is in a class that is strictly lower than DLOGSPACE and does not contain the parity test, then we would have solved Conjecture 1.

It is known that $\text{AC}^0 \subset$ DLOGSPACE [5, 27]. If $\mathcal{NRL}^{\text{aggr}}$ had AC^0 data complexity, the same argument would solve at least Conjectures 1 and 2. However, while queries written in $\mathcal{NRL}$ have AC^0 data complexity [51], it is not hard to see that there are non-AC^0 queries in $\mathcal{NRL}^{\text{aggr}}$ since multiplication is not in AC^0 [5]. As a more interesting example, recall that parity of cardinality is definable in $\mathcal{NRL}^{\text{aggr}}$ if $\mathbb{Q}$, *unit* and *bool* are the only base types. Note that this does not mean Conjecture 1 is wrong. Conjecture 1 asks, in particular, if the parity of the cardinality of a set of elements of an *unordered* base type is definable in $\mathcal{NRL}^{\text{aggr}}$. The method above cannot answer this question. It only shows that there exist non-AC^0 queries definable in $\mathcal{NRL}^{\text{aggr}}$.

A simple complexity argument does not help us, nor do we know if the bounded degree property holds for $\mathcal{NRL}^{\text{aggr}}$, so we use another technique to prove the desired results. It is well known that properties of cardinalities of finite models which can be tested in the first-order logic are either finite or co-finite. We proved a similar result in Section 5.2 in the course of investigating the arithmetic power of $\mathcal{BQL}$. Now, using the conservative extension property, we present two results of the same kind for $\mathcal{NRL}^{\text{aggr}}$. We show that for certain families of graphs a similar finiteness-cofiniteness property holds. Then we derive the inexpressibility of *chain* and *bbtree* in $\mathcal{NRL}^{\text{aggr}}$. Since $\mathcal{BQL}$ can be embedded in $\mathcal{NRL}^{\text{aggr}}$, these results confirm Conjecture 1, Conjecture 2, and the second part of Conjecture 3.

The first family of graphs to be considered are the k-multi-cycles. A binary relation $O : \{b \times b\}$ is called a *k-multi-cycle* if it is nonempty and is of the form shown in Figure 4 where $h \geq k$ and o_i^j are all distinct. That is, it is a graph containing $m \geq 1$ unconnected cycles of equal length $h \geq k$. Here b is an uninterpreted base type with countably infinite domain on which only the equality test is available.

Theorem 5.15 *Let $G : \{b \times b\} \to bool$ be a function expressible in $\mathcal{NRL}^{\text{aggr}}$. Then there is some k such that for all k-multi-cycles O, it is the case that $G(O)$ is true; or for all k-multi-cycles O, it is the case that $G(O)$ is false.*

Let us observe that if we identify isomorphic k-multi-cycles, then for any $m \geq 1$, Theorem 5.15 says that $\mathcal{NRL}^{\text{aggr}}$-definable properties of k-multi-cycles consisting of at most m components are either finite or co-finite. Hence $\mathcal{NRL}^{\text{aggr}}$ cannot distinguish one k-multi-cycle from another as long as the cycles are long enough. In $\mathcal{NRL}^{\text{aggr}}(chain)$ it is possible to distinguish k-multi-cycles containing one cycle from those containing two. Therefore, *chain* is not expressible in $\mathcal{NRL}^{\mathrm{aggr}}$. These two observations together with the fact that $\mathcal{BQL}$ can be embedded in $\mathcal{NRL}^{\mathrm{aggr}}$ and Corollary 5.11 settle Conjectures 1 and 2.

Corollary 5.16 *Parity test on cardinality of relations is not expressible in $\mathcal{NRL}^{\mathrm{aggr}}$ and hence not in $\mathcal{BQL}$.* □

Corollary 5.17 *Transitive closure of binary relations is not expressible in $\mathcal{NRL}^{\mathrm{aggr}}$ and hence not in $\mathcal{BQL}$.* □

The second family of graphs to be considered are the k-strict-binary-trees. A k-strict-binary-tree is a nonempty tree where each node has either 0 or 2 descendents and the distance from the root to any leaf is at least k.

Theorem 5.18 *Let $G : \{b \times b\} \to bool$ be a function expressible in $\mathcal{NRL}^{\mathrm{aggr}}$. Then there is some k such that for all k-strict-binary-trees O, it is the case that $G(O)$ is true; or for all k-strict-binary-trees O, it is the case that $G(O)$ is false.*

The immediate consequence of this theorem is that $\mathcal{NRL}^{\mathrm{aggr}}$ cannot distinguish one k-strict-binary-tree from another as long as the trees are deep enough. In $\mathcal{NRL}^{\mathrm{aggr}}(bbtree)$, for any $k > 0$, one can distinguish a balanced binary tree of height k from any other k-strict-binary-tree. Therefore, we have settled Conjecture 3.

Corollary 5.19 *The test for balanced binary trees is not definable in $\mathcal{NRL}^{\mathrm{aggr}}$ and hence not in $\mathcal{BQL}$.* □

In summary,

Corollary 5.20 *All the queries listed in Corollary 5.11 are not expressible in $\mathcal{NRL}^{\mathrm{aggr}}$.* □

In contrast to the inexpressibility result of Consens and Mendelzon [15], which depends on the separation of DLOGSPACE and NLOGSPACE, our results do not have such preconditions.

Let us make another observation before proving the main theorems. Some of the problems considered above are known to be complete for various complexity classes under first-order reductions. For example, the graph reachability problem is first-order complete for NLOGSPACE and its restriction to graphs with outdegree 1 is first-order complete for DLOGSPACE. Using the results of Immerman [24] on first-order completeness, the fact that $\mathcal{NRL}$ and $\mathcal{NRL}^{\mathrm{aggr}}$ are closed under first-order reductions (the proof of this is similar to Immerman's [25]), and the inexpressibility results proved in this paper, we get

Corollary 5.21 *Let P be a problem that is complete with respect to first-order reductions for one of the following classes: DLOGSPACE, Sym-LOGSPACE, NLOGSPACE, PTIME. Then P cannot be solved by $\mathcal{NRL}^{\mathrm{aggr}}$ or by $\mathcal{BQL}$.* □

In the remainder of this section, we prove the main theorems. As the technique applied is sophisticated, we first present the "eureka" step before we present the proof details.

Since graph queries have height 1, by the conservative extension property of $\mathcal{NRL}^{\text{aggr}}$, it is only necessary to consider expressions having height 1. Observe further that the normal forms produced by the rewriting done in the conservative extension theorem have a rather special trait. Let $e : \mathbb{Q}$ be an expression of height 1 in normal form. Let $R : \{b \times b\}$ be the only free variable in e. Let b be an unordered base type. Let e contain no constant of type b. Then e contains no subexpression of the form $\bigcup\{e_1 \mid x \in e_2\}$. Also, every subexpression involving $\sum$ is guaranteed to have the form $\sum \{\!|e_1 \mid x \in R|\!\}$.

It is natural to speculate on what e can look like. The most natural shape that comes to mind is the one depicted below.

$$\sum\left\{\!\!\left|\ \ldots \sum\left\{\!\!\left| \begin{array}{l} \textit{if } P_1 \\ \textit{then } f_1 \\ \vdots \\ \textit{else if } P_h \\ \textit{then } f_h \\ \textit{else } f_{h+1} \end{array} \right| x_1 \in R \right|\!\!\right\} \ldots \right| x_n \in R \right|\!\!\right\}$$

Assume that the probability, in terms of the number of edges in R, of P_i being true and $P_{j<i}$ being false is p_i. Then the expression above is equivalent to the polynomial $N^n \cdot (p_1 \cdot f_1 + \ldots + p_{h+1} \cdot f_{h+1})$, with N being the number of edges in R.

This observation is crucial for two reasons. First, the use of the summation operator is no longer arbitrary. It is now used only for computing the number of edges in R. All other uses of it have been replaced by a polynomial expression. Second, the expression no longer depends on the topology of the graph R. The only thing in R that can affect the value of the polynomial (and hence the original expression) is the cardinality of R. This leads to finite-cofiniteness of graph queries for which the probability assumption holds.

The insight above leads to a search for classes of graphs that possess sufficient regularity so that the required probability analysis can be performed. The simplest class of such graphs is perhaps the k-multi-cycles. We first present two preliminary definitions and demonstrate the probability analysis on k-multi-cycles. The proofs of Theorem 5.15 and Theorem 5.18 are then sketched.

Define $\textit{distance}_c(o, o', O)$ to be a predicate that holds iff the distance from node $\pi_1 o$ to node $\pi_2 o'$ in graph O is c. Note that $\textit{distance}_c$ is definable in $\mathcal{NRL}^{\text{aggr}}$ for each constant c.

Define a *d-state* S with respect to variables $R : \{b \times b\}$, x_1, ..., $x_m : b \times b$ to be a conjunction of formulae of the form $\textit{distance}_c(x_i, x_j, R)$ or the form $\neg\textit{distance}_c(x_i, x_j, R)$, such that for each $0 \leq c \leq d$, $1 \leq i, j \leq m$, either $\textit{distance}_c(x_i, x_j, R)$ or $\neg\textit{distance}_c(x_i, x_j, R)$ must appear in the conjunction. Also S has to be satisfiable in the sense that some chain O and edges o_1, ..., o_m in O can be found so that $S[O/R, o_1/x_1, ..., o_m/x_m]$ holds.

Proposition 5.22 *Let e be an expression of $\mathcal{NRL}^{\text{aggr}}$ having $R : \{b \times b\}$, $N : \mathbb{Q}$, x_1, ..., $x_m : b \times b$ as* *free variables such that e has the special form below*

$$\sum\left\{\left|\ \ldots \sum\left\{\left|\ \begin{array}{l}\textit{if } P\\ \textit{then } E\\ \textit{else } 0\end{array}\ \right|\ x_{m+1}\in R\ \right|\right\}\ \ldots\ \left|\ x_{m+n}\in R\ \right|\right\}$$

where E is a ratio of polynomials in terms of N, P is a boolean combination of formulae of the form $\pi_i x_{i'} = \pi_j x_{j'}$, $\pi_i x_{i'} \neq \pi_j x_{j'}$, $\neg distance_c(x_i, x_j, R)$, or $distance_c(x_i, x_j, R)$. Then there is a constant D such that for any $d > D$ and any d-state S with respect to R, x_1, ..., x_m; there is a ratio r of polynomials in terms of N such that for any d-multi-cycle O and any edges o_1, ..., o_m in O making $S[O/R, o_1/_1, \ldots, o_m/x_m]$ true, it is the case that $e[O/R, o_1/x_1, \ldots, o_m/x_m, card(O)/N] = r[card(O)/N]$.

Proof. The constant D can be chosen as any number that is not less than the longest separation between any nodes $\pi_1 x_i$ and $\pi_2 x_j$ in any graph R described by P that is dictated by P to be connected. The constant D should also be larger than the minimum separation between any nodes $\pi_1 x_i$ and $\pi_2 x_j$ in any graph R described by P that is not dictated by P to be connected.

This number can be estimated as follows. Assume, without loss of generality, that P is $Q_1 \vee \ldots \vee Q_n$, where each Q_i contains only conjunctions. Let d_i be the sum of the c's for each $distance_c(x_i, x_j, R)$ or $\neg distance_c(x_i, x_j, R)$ in Q_i. Let D be the maximum of these d_i's plus $m+n$. The $m+n$ is added because an item of the form $\pi_2 x_i = \pi_1 x'_j$ is equivalent to $distance_1(x_i, x_j, R)$. An easy upper bound for D is $(n+m)\cdot(C+1)$, where C is the sum of the c's for each $distance_c(x_i, x_j, R)$ or $\neg distance_c(x_i, x_j, R)$ in P.

By the probability p for a predicate P of n free variables to hold with respect to a graph O, we mean the proportion of the instantiations of the free variables to edges in O that make P true. The key to this proposition is in realizing that the probability p for P to hold can be determined in the case of k-multi-cycle when k is large (any $k > D$ is good enough). Moreover p can be expressed as a ratio of two polynomials of N. Thus r can be defined as $N^n \cdot p \cdot E$.

The probability p can be calculated as follows. First, given $d > D$, we generate all possible d-states D_j's with respect to the variables R, x_1, ..., x_{m+n}. Second, determine the probability q_j of D_j given the certainty of S; this can be calculated using the procedure to be given shortly. Third, eliminate those D_j's that are inconsistent with the conjunction of S and P. (Note that a D_j that is consistent with S can only be inconsistent with P in two ways: P is already inconsistent or P demands R to contain a cycle of length shorter than D. The proposition restricts our attention to cycles of length $d > D$; hence dictates the elimination of the second kind of inconsistency above. By picking the D to be longer than any cycles that can be mentioned in P as we have done, we have essentially restricted P to a predicate that does not mention cycles, thus implying the probability analysis below.) Finally, calculate p by summing the q_j's corresponding to those remaining d-states.

It remains to show that each q_i can be expressed as a ratio of two polynomials in N. Partition the positive atomic formulae of the corresponding D_i into groups so that the variables in each group are connected between themselves and are unconnected with those in other groups. (Variables x and y are said to be connected in D_i if there is a positive atom $distance_c(x, y, R)$ in D_i.) Note that the negative atomic formulae merely assert that these groups are unconnected. Then we proceed by induction on the number of groups.

The base case is when there is just one group. In such a situation, all the variables lie on the same cycle. Since a d-state can be satisfied by a chain of length d, these variables must lie on a line. Let u be the number of bound variables amongst x_{m+1}, ..., x_{m+n} appearing in the group; in this case $u = n$. Then $q_i = N \div N^u$ if no variables amongst x_1, ..., x_m appear in the group. Otherwise, $q_i = 1 \div N^u$. In either case, q_i is a ratio of polynomials in N.

For the induction case, suppose we have more than one group. The independent probability of each group can be calculated as in the base case. Then q_i is the difference between the product of these independent probabilities and the sum of the probabilities where these groups are made to overlap in all possible ways. These groups are made to overlap by turning some negative leaves in D_i into positive ones so that the results are again d-states. Notice that when groups overlap, the number of groups strictly decreases. Hence the induction hypothesis can be applied to obtain these probabilities as ratios of polynomials in N. Consequently, q_i can be expressed as a ratio of polynomials in N as desired. $\square$

Having established the above key result, Theorem 5.15 can be proved as follow.

Proof sketch of Theorem 5.15. Let $G : \{b \times b\} \to \mathit{bool}$ be implemented by the $\mathcal{NRL}^{\mathrm{aggr}}$ expression $\lambda R.E$. Without loss of generality, E can be assumed to be a normal form with respect to the rewrite system used in the proof of Theorem 5.1. We note that such an E contains no subexpression of the form $\bigcup\{e_1 \mid x \in e_2\}$. Furthermore, all occurrences of summation in E must be of the form $\sum \{\!|e \mid x \in R|\!\}$. These two observations on E come directly from the rewrite rules used in Theorem 5.1. For example, the e_2 in $\sum \{\!|e_1 \mid x \in e_2|\!\}$ is always simplified by one of these rules, unless it is already a variable.

Let us temporarily enrich the language with the usual logical operators $\vee$, $\wedge$, $\neg$, $\neq$, $\not\leq$, as well as $\mathit{distance}_c$ and $\neg\mathit{distance}_c$. Also introduce a new variable $N : \mathbb{Q}$, which is to be interpreted as the cardinality of R. Rewrite all summations into the special form given below

$$\sum \left\{\!\!\left| \cdots \sum \left\{\!\!\left| \begin{array}{l} \mathit{if}\ P \\ \mathit{then}\ f \\ \mathit{else}\ 0 \end{array} \;\middle|\; x_{m+1} \in R \right|\!\!\right\} \cdots \;\middle|\; x_{m+n} \in R \right|\!\!\right\}$$

so that f has the form $h \div g$, where h is a polynomial in N and g is either a polynomial in N or is again a subexpression of the same special form. Also, P is a boolean combination of formulae of the following form: $\pi_i x_{i'} = \pi_j x_{j'}$, $\pi_i x_{i'} \neq \pi_j x_{j'}$, $\mathit{distance}_c(x_i, x_j, R)$, $\neg\mathit{distance}_c(x_i, x_j, R)$, $U =^{\mathbb{Q}} V$, $U \neq^{\mathbb{Q}} V$, $U \leq V$, or $U \not\leq V$, where U and V also have the same special form.

Let the resultant expression be F. The rewriting should be such that for all sufficiently long k-multi-cycles O, $F[O/R, \mathit{card}(O)/N]$ holds if and only if $E[O/R]$ holds. This rewriting can be accomplished by using rules such as

- $\mathit{if}\ e_1\ \mathit{then}\ \sum \{\!|e_2 \mid x \in R|\!\}\ \mathit{else}\ e_3 \leadsto \sum \{\!|\ \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \div N \mid x \in R|\!\}$
- $\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ \sum \{\!|e_3 \mid x \in R|\!\} \leadsto \sum \{\!|\ \mathit{if}\ e_1\ \mathit{then}\ e_2 \div N\ \mathit{else}\ e_3 \mid x \in R|\!\}$
- $e_1 \cdot \sum \{\!|e_2 \mid x \in R|\!\} \leadsto \sum \{\!|e_1 \cdot e_2 \mid x \in R|\!\}$
- $\sum \{\!|e_1 \mid x \in R|\!\} \cdot e_2 \leadsto \sum \{\!|e_1 \cdot e_2 \mid x \in R|\!\}$
- $\sum \{\!|e_1 \mid x \in R|\!\} \div e_2 \leadsto \sum \{\!|e_1 \div e_2 \mid x \in R|\!\}$

- $\sum \{\!|e_1 \mid x \in R|\!\} + e_2 \rightsquigarrow \sum \{\!|e_1 + (e_2 \div N) \mid x \in R|\!\}$
- $\sum \{\!|e_1 \mid x \in R|\!\} - e_2 \rightsquigarrow \sum \{\!|e_1 - (e_2 \div N) \mid x \in R|\!\}$
- $e_1 - \sum \{\!|e_2 \mid x \in R|\!\} \rightsquigarrow \sum \{\!|(e_1 \div N) - e_2 \mid x \in R|\!\}$
- $e_1 + \sum \{\!|e_2 \mid x \in R|\!\} \rightsquigarrow \sum \{\!|(e_1 \div N) + e_2 \mid x \in R|\!\}$
- $\sum \{\!|\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \mid x \in R|\!\} \rightsquigarrow \sum \{\!|\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ 0 \mid x \in R|\!\} + \sum \{\!|\mathit{if}\ \neg e_1\ \mathit{then}\ e_3\ \mathit{else}\ 0 \mid x \in R|\!\}$, if neither e_2 nor e_3 is 0.

We do not need a rule for rewriting $e_1 \div \sum \{\!|e_2 \mid x \in R|\!\}$ when e_1 is not a summation because it is already of the right form. Having obtained F in this special form, the proof is continued by repeating the following steps until all occurrences of R have been eliminated.

Step 1. Look for an innermost subexpression of F that has the special form required by Proposition 5.22. Let this subexpression be F' and its free variables be y_1, ..., y_m, R and N. Generate all possible d-states with respect to these free variables of F'. The d is the smallest one suggested by Proposition 5.22 and serves as a lower bound for k. Let S_1, ..., S_{h+1} be these d-states. Apply Proposition 5.22 to F' with respect to each S_i to obtain expressions r_i which are ratios of polynomials of N. Then F' is equivalent to $\mathit{if}\ S_1\ \mathit{then}\ r_1\ \mathit{else}\ \ldots \mathit{if}\ S_h\ \mathit{then}\ r_h\ \mathit{else}\ r_{h+1}$ under the assumption of the theorem that the variable R is never instantiated to short k'-multi-cycles where $k' < k$.

Step 2. To maintain the same special form, we need to push the S_i up one level to the expression in which F' is nested. This rewriting is done using rather simple rules:

- $(\mathit{if}\ S_1\ \mathit{then}\ r_1\ \ldots \mathit{if}\ S_h\ \mathit{then}\ r_h\ \mathit{else}\ r_{h+1}) =^{\mathbb{Q}} V \rightsquigarrow (S_1 \wedge r_1 =^{\mathbb{Q}} V) \vee \cdots \vee (S_{h+1} \wedge r_{h+1} =^{\mathbb{Q}} V)$
- $\mathit{if}\ P\ \mathit{then}\ (f \div (\mathit{if}\ S_1\ \mathit{then}\ r_1\ \mathit{else}\ \ldots \mathit{if}\ S_h\ \mathit{then}\ r_h\ \mathit{else}\ r_{h+1}))\ \mathit{else}\ e$
  $\rightsquigarrow \mathit{if}\ P \wedge S_1\ \mathit{then}\ f \div r_1\ \ldots \mathit{if}\ P \wedge S_{h+1}\ \mathit{then}\ f \div r_{h+1}\ \mathit{else}\ e$

Step 3. After Step 2, some expression having the form $U =^{\mathbb{Q}} V$, $U \leq V$, or their negation, can become an equation of ratios of polynomials of N. Such an expression can be replaced either by *true* or by *false*. For illustration, we explain the case of $U =^{\mathbb{Q}} V$; the other cases are similar. First, $U =^{\mathbb{Q}} V$ is readily transformed into a polynomial $P = 0$ with N being its only free variable. Check if P is identically 0. In that case, replace $U =^{\mathbb{Q}} V$ by *true*. If P is not identically 0, we use the fact that a polynomial has a finite number of roots. By choosing a sufficiently large lower bound for k, we can ensure that N always exceeds the largest root of P. Thus, in this case we replace $U =^{\mathbb{Q}} V$ by *false*.

Observe that in Step 1 we have reduced the number of summations and in Step 3 we have reduced the number of equality and inequality tests. By repeating these steps, we eventually reach the base case and arrive at an expression where R does not occur. When we are finished, the resultant expression is clearly a boolean formula containing no free variables. Therefore its value does not depend on R. Consequently the theorem holds for any k not smaller than the lower bound determined by the above process. □

The proof of Theorem 5.15 relies on two things: satisfiability of d-states is easy to decide for k-multi-cycles and probabilities are easy to calculate and express as ratios of polynomials in terms of the size of graphs for k-multi-cycles. These two properties are also enjoyed by k-strict-binary-trees.

Proof sketch of Theorem 5.18. It is easy to decide if a d-state is satisfiable by some k-strict-binary-trees. The probability calculation is also simple. The only problem is that the probability must be expressed as a ratio of polynomials of the number of edges in the tree. This is dealt with by observing that in k-strict-binary-trees, the number of internal nodes is 1 less than half the number of edges and the number of leaves is equal to 2 plus the number of internal nodes. The theorem follows by repeating verbatim the proof for k-multi-cycles. □

6 Power Operators, Bounded Loop, and Structural Recursion

In the previous section, we saw that $\mathcal{BQL}$ and $\mathcal{NRL}^{\text{aggr}}$ have the same limitations as most languages based on first-order logic: they cannot express recursive queries. There are several approaches to adding expressive power to set languages. In this section, we study three of them for $\mathcal{BQL}$ and $\mathcal{NRL}^{\text{aggr}}$.

Abiteboul and Beeri [1], as well as Gyssens and Van Gucht [22], used *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 [6] introduced structural recursion as an alternative means for increasing the horsepower of query languages.

It was shown in Tannen et al. [8], see also Gyssens and Van Gucht [22], that endowing $\mathcal{NRL}(eq)$ with a structural recursion primitive 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.

We have been using structural recursion on the union presentation. We also mentioned in the beginning that sets can be equivalently constructed by starting with empty set and inserting new elements. There is a corresponding structural recursion construct, called *s_sri* for *structural recursion on the insert presentation*. It is known to have precisely the same power as *s_sru* [6], and it is sometimes easier to use. The syntax for this construct 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 [7]. 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 [6] 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 [7]. 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 the union of the 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. The intersection of those elements is the transitive closure of R. Moreover, Paredaens and Suciu showed [52] that any algorithm for computing transitive closure in $\mathcal{NRL}(powerset)$, evaluated under the standard operational semantics, must use exponential space. Even though different evaluation schemes proposed recently [2, 33] give polynomial space algorithms for transitive closure in the powerset algebra, it is conjectured that no reasonable evaluation strategy will give us polynomial time algorithms.

We do not advocate the elimination of every expensive operation 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. We show that these non-polynomial bag operators are strictly more expressive than their set analogs. Furthermore, we prove that the analog of the *gen* primitive on sets fills the gap. We also characterize the arithmetic expressive power of bag languages endowed with power operators and structural recursion. In particular, we prove that they define precisely the classes of elementary and primitive recursive functions.

6.1 Powerset, Powerbag, and Structural Recursion

Grumbach and Milo [21], 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,

$$powerset\{\!|1,1,2|\!\} \;=\; \{\!|\{\!||\!\},\{\!|1|\!\},\{\!|2|\!\},\{\!|1,1|\!\},\{\!|1,2|\!\},\{\!|1,1,2|\!\}|\!\}$$

We do not consider *powerset* on bags further because

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

Proof. We have to show how to express *powerbag* given *powerset*. 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 $count(a,B)$. B' can be constructed in $\mathcal{BQL}(powerset)$ because selection is definable. Let $B'' = unique(B')$. Now observe that replacing the second component of every pair by its *powerset* and then $map(b\_\rho_2)$ followed by flattening gives us a bag where each element $a \in B$ is given a unique label. 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 cannot be automatically verified [7]. Its semantics is similar to the semantics of s_sri. We want to show that *powerbag* is strictly weaker than b_sri.

Let *hyper* be the hyper-exponentiation function. That is, $hyper(0, n) = n$ and $hyper(m + 1, n) = 2^{hyper(m,n)}$. In other words, $hyper(m, n)$ is a stack of m 2's with n at the top. Define $\mathsf{size}\, o$, the size of object o, as follows: it is 1 for objects of base types, sum of the sizes of the components for pairs and sum of the sizes of the elements for bag type. Then

Proposition 6.2 *Let $f : s \to t$ be an expression of $\mathcal{BQL}(powerbag)$. Then there exists a constant c_f such that for every object $o : s$, $\mathsf{size}\, f(o) \leq hyper(c_f, \mathsf{size}\ o)$.*

Proof sketch. The proof is by induction on the structure of f. For any polynomial operator p in $\mathcal{BQL}$, it is safe to define c_p to be 1. For operators that are not polynomial, define $c_{powerbag} := 1$, $c_{\langle f,g \rangle} := \max(c_f, c_g)$, $c_{f \circ g} := c_f + c_g$, and $c_{b\_map(f)} := 1 + c_f$. □

The above establishes an upper bound on the size of output of queries in $\mathcal{BQL}(powerbag)$. This upper bound is later used to characterize the arithmetic properties of $\mathcal{BQL}(powerbag)$. But its immediate consequence is the separation of *powerbag* from b_sri.

Theorem 6.3 $\mathcal{BQL}(powerbag) \subset \mathcal{BQL}(b\_sri)$.

Proof. Inclusion is easy [8]. To prove strictness, define an auxiliary function $g : \{\!|unit|\!\} \to \{\!|unit|\!\}$ in $\mathcal{BQL}(b\_sri)$ by $g := b\_map(!) \circ powerbag$. It is easy to see that on an input of size n, g produces the output of size 2^n. Now define $f := \lambda n.sri(g \circ \pi_2, n)(n)$. A straightforward analysis shows that $\mathsf{size}\, f(o) = hyper(\mathsf{size}\ o, \mathsf{size}\ o)$. Therefore, by Proposition 6.2, f cannot be expressed in $\mathcal{BQL}(powerbag)$. □

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 [6] but its well-definedness precondition cannot be verified by a compiler [7]. 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. 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 $\mathcal{BQL}(loop) \simeq \mathcal{BQL}(b\_sri)$.

Proof. For the $\mathcal{BQL}(loop) \subseteq \mathcal{BQL}(b\_sri)$ part, it suffices to observe that $loop(f)(n, e) = b\_sri(f \circ \pi_2,\ e)(n)$. This part was also proved by Saraiya [48]. The $\mathcal{BQL}(b\_sri) \subseteq \mathcal{BQL}(loop)$ part is more involved. Let

$$\Phi_f(R) \ := \ \{\!|(A \ monus \ \{\!|a|\!\},\ f(a, b)) \mid (A, b) \in R,\ a \in A|\!\}$$

Should the f above fail the commutativity requirement, $b\_map(\pi_2)(unique(loop(\Phi_f)(n, \{\!|(n, e)|\!\}))$ is then a bag containing all possible outcomes (one for each order of applying f) of $b\_sri(f, e)(n)$. However, if $f : s \times t \to t$ satisfies the commutativity precondition, then $unique(loop(\Phi_f)(n, \{\!|(n, e)|\!\}))$ is a singleton bag and is equal to $\{\!|(\{\!||\!\}, b\_sri(f, e)(n))|\!\}$. $b\_map(\pi_2)$ can then be applied to the result to get a singleton bag containing $b\_sri(f, e)(n)$. This shows that b_sri is expressible in $\mathcal{BQL}(loop)$. $\square$

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 can pick a particular order of application of i in $b\_sri(i, e)$, then more efficient implementations are possible. For example, define $\Phi'_f(R)$ as $\{\!|(A \ monus \ \{\!|a|\!\}, f(\pi_2\, a, b)) \mid (A, b) \in R, a \in unique(\max(A))|\!\}$, where max returns the subbag of maximal elements with respect to the linear order (see Proposition 5.2). Then $loop(\Phi'_f)(X, \{\!|(sort(X), e)|\!\})$ returns $\{\!|(\{\!||\!\}, b\_sri(f, e)(X))|\!\}$. However, if f is not commutative, then $loop(\Phi'_f)(X, \{\!|(sort(X), e)|\!\})$ equals to $\{\!|(\{\!||\!\}, f(o_1, f(o_2, f(\ldots, f(o_k, e) \ldots)))))|\!\}$ where $X = \{\!|o_1, \ldots, o_k|\!\}$ and $o_1 \leq \ldots \leq o_k$ is the linear order of Proposition 5.2.

6.3 Arithmetic Properties of Non-Polynomial Languages

In this section we characterize the arithmetic expressive power of $\mathcal{BQL}(powerbag)$ and $\mathcal{BQL}(loop)$. Before proving the two theorems, let us argue that they are very intuitive and are not unexpected. Recall two classical results in recursion theory [41]. One, due to Meyer and Ritchie, states that the functions computable by the language that has assignment statement and *for n do S*, are precisely the primitive recursive functions. The semantics of *for n do S* is to repeat S n times. A similar result by Robinson, later improved by Gladstone, says that the primitive recursive functions are functions built from the initial functions by composition and iteration. That is, $f(n, \vec{x}) = g^{(n)}(\vec{x})$; see Odifreddi [41]. In view of these results and the fact the *loop* construct is just a *for–do* iteration, the following result is very natural.

Theorem 6.5 *The class of functions* $f : \mathbb{N} \times \ldots \times \mathbb{N} \to \mathbb{N}$ *definable in* $\mathcal{BQL}(loop)$ *coincides with the class of primitive recursive functions.*

Grumbach and Milo [21] showed that their bag language, which is equivalent to $\mathcal{BQL}(powerbag)$, expresses all elementary queries. They obtained this result by encoding computations on Turing machines in the language. Recall that the class of Kalmar-elementary functions $\mathcal{E}$ is the smallest class that contains basic functions, addition, multiplication, modified subtraction $\dot{-}$ and is closed under bounded sums and bounded products [47]. That is, the following functions are in $\mathcal{E}$ if g is in $\mathcal{E}$:

$$f_1(n, \vec{x}) = \sum_{i=0}^{n} g(i, \vec{x}) \qquad\qquad f_2(n, \vec{x}) = \prod_{i=0}^{n} g(i, \vec{x})$$

Using different techniques, we prove the following:

Theorem 6.6 *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.*

Let us first give the

Proof of Theorem 6.5. Throughout the proof we use $\mathbb{N}$ as abbreviation for $\{\!|unit|\!\}$ and n as an abbreviation for $\{\!|(), \ldots, ()|\!\}$ (n times). First observe that since *powerbag* can be expressed in the language, 2^n as a function of n can be expressed as we have done it in the proof of Theorem 6.3. Therefore, encoding and decoding functions for tuples can be expressed. In view of that and the Robinson-Gladstone result [41], to prove that all primitive recursive functions can be computed by $\mathcal{BQL}(loop)$, it is enough to show that if $g(m)$ can be computed, then so can $f(n, m) = g^{(n)}(m)$. But this is obvious because $f(n, m) = loop(g)(n, m)$.

To prove the converse, first verify this claim: for any expression $e : s \to t$ in $\mathcal{BQL}(loop)$ there is a monotone primitive recursive function φ_e of one argument such that $\mathsf{size}\, e(o) \le \varphi_e(\mathsf{size}\, o)$. The verification proceeds by structural induction on e. The only two problematic cases are $b\_map(f)$ and $loop(f)$. Let $f : s' \to t'$ and $b\_map(f)(d) = d'$ where $d = \{\!|o_1, \ldots, o_k|\!\}$ and $d' = \{\!|o'_1, \ldots, o'_k|\!\}$. Then $\mathsf{size}\, d' = \sum_i \mathsf{size}\, o'_i \le \sum_i \varphi_f(\mathsf{size}\, o_i) \le \sum_i \varphi_f(\mathsf{size}\, d) \le \mathsf{size}\, d \cdot \varphi_f(\mathsf{size}\, d)$. So $\varphi_{b\_map(f)}$ can be picked to be $n \cdot \varphi_f(n)$ which is clearly monotonic. For the case of $loop(f)$, define $\varphi_{loop(f)}(n) = \varphi_f^{(n)}(n)$. From monotonicity of φ_f it can be easily derived that $\varphi_{loop(f)}$ satisfies the desired property and is monotone itself.

Now a straightforward translation of the operations of $\mathcal{BQL}(loop)$ into computations on a Turing machine and the observation we have just made show that the space complexity for every expression in the language remains bounded by a primitive recursive function. Therefore, if $f : \mathbb{N} \times \ldots \times \mathbb{N} \to \mathbb{N}$

is a function computable by $\mathcal{BQL}(loop)$, it is recursive and the space complexity (and therefore time complexity) of its computation on a Turing machine is bounded by a primitive recursive function. Now, if f is obtained from the initial functions by using primitive recursion schema and minimization, this shows that every instance of minimization can be replaced by bounded minimization which is known not to enlarge the class of primitive recursive functions. Thus, f is primitive recursive. This completes the proof. $\square$

Next we give the

Proof of Theorem 6.6. First we show that bounded sum and bounded product are expressible in $\mathcal{BQL}(powerbag)$. Since coding functions for tuples are available, we restrict ourselves only to the case of $f_1(n) = \sum_{i=0}^{n} g(i)$ and $f_2(n) = \prod_{i=0}^{n} g(i)$. Let $powerset := unique \circ powerbag$. It is easy to see that $powerset(n) = \{\!|0, 1, 2, \ldots, n|\!\}$. Therefore, $b\_\mu \circ b\_map(g)$ applied to $powerset(n)$ gives us $f_1(n)$. The proof of the expressibility of f_2 resembles the proof of the expressibility of the α primitive of [34] for or-sets. Again, g is mapped over $powerset(n)$ to obtain $\{\!|g(0), g(1), \ldots, g(n)|\!\}$. If at least one of $g(i)$ is 0 (that is, an empty bag), the result is 0. Otherwise each occurrence of () inside each $g(i)$ is paired with i. The resulting bag is flattened and the powerbag is taken. From this powerbag such subbags are selected that they contain exactly one pair tagged with i for each i. The number of such subbags is exactly $f_2(n)$. So f_2 is expressible.

The proof of the converse is similar to the proof for the primitive recursive functions. The space complexity for every expression in $\mathcal{BQL}(powerbag)$ is bounded above by $hyper(c, n)$ where c is a constant, see Proposition 6.2. That is, by a function in $\mathcal{E}$. Again, a simple translation into computation on a Turing machine shows that complexity remains bounded by a function from $\mathcal{E}$. Now if $f : \mathbb{N} \times \ldots \times \mathbb{N} \to \mathbb{N}$ is computable in $\mathcal{BQL}(powerbag)$, it can be computed by a Turing machine whose space complexity is bounded by a function from $\mathcal{E}$. Whence $f \in \mathcal{E}$; see Machtey and Young [39]. This finishes the proof of Theorem 6.6. $\square$

As a corollary, we show how to obtain all *unary* primitive recursive functions using simpler constructs. First observe that $powerset^{unit} : \{\!|unit|\!\} \to \{\!|\{\!|unit|\!\}|\!\}$ is a polynomial operation: $powerset^{unit}(n) = \{\!|0, 1, 2, \ldots, n|\!\}$. We simplify the loop construct by defining $iter(f) : \{\!|t|\!\} \to \{\!|unit|\!\}$ where $f : \{\!|unit|\!\} \to \{\!|unit|\!\}$ by $iter(f)\{\!|o_1, \ldots, o_n|\!\} = f(f(\ldots(f\{\!||\!\})\ldots))$ where is f applied n times.

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

Proof. It is known that all unary primitive recursive functions can be obtained from the iteration schema: $g(n) = f^{(n)}(0)$ and an extended list of initial functions, see Rose [47, Theorem 1.4]. It is straightforward to verify that all additional initial functions can be expressed in the presence of $powerset^{unit}$. $\square$

6.4 Power Operators and Structural Recursion on Sets and Bags

We have introduced power operators and structural recursion for sets and bags. We also know that $\mathcal{BQL} \simeq \mathcal{NRL}^{\mathrm{nat}}$. Under the translations of Theorem 4.4, $n : \mathbb{N}$ is carried to a bag of n units: $\{\!|(), \ldots, ()|\!\}$. Consider the following primitive from Corollary 5.9:

$$gen : \mathbb{N} \to \{\mathbb{N}\}, \qquad gen(n) = \{0, 1, \ldots, n\}$$

Under the translations of Theorem 4.4, it corresponds to the bag language primitive that takes a bag of n units and returns the 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 6.8
- $\mathcal{NRL}^{\rm nat}(powerset) \subset \mathcal{BQL}(powerbag)$.
- $\mathcal{NRL}^{\rm nat}(s\_sri) \subset \mathcal{BQL}(b\_sri)$.

Proof. Inclusion easily follows from Theorem 4.4. We only have to demonstrate that $powerset : \{s\} \to \{\{s\}\}$ is definable using *powerbag* and the translations of Theorem 4.4. This is indeed the case because, taking a set X, translating it into a bag, applying $unique \circ powerbag$ to it, translating it back to sets and projecting out multiplicities we obtain the powerset of X.

To show strictness, observe that $powerset^{unit}$ is definable in both $\mathcal{BQL}(powerbag)$ and $\mathcal{BQL}(b\_sri)$. Hence, in view of Theorem 6.3, it is enough to show that *gen* is not expressible in $\mathcal{NRL}^{\rm nat}(s\_sri)$. 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 $\mathcal{NRL}^{\rm nat}(s\_sri)$ there exists a monotone primitive recursive function φ_f such that, if $f(i) = o$ and sizes of i and o are s_i and s_o, then $s_o \le \varphi_f(s_i)$. To show this, use structural induction on expressions of $\mathcal{NRL}^{\rm nat}(s\_sri)$. For operations other than *s_sri* we can use φ as defined in the proof of Theorem 6.5. For $s\_sri(e, g)$ we define:

$$\begin{array}{rcl} \varphi_{s\_sri(e,g)}(0) & = & size(e) \\ \varphi_{s\_sri(e,g)}(n) & = & \varphi_g(n + \varphi_{s\_sri(e,g)}(n-1)) \end{array}$$

It can be seen that the inductive assumption of monotonicity of φ_g implies that $\varphi_{s\_sri(e,g)}$ satisfies the desired property. Now assume that *gen* is definable. Let $n = \varphi_{gen}(1)$. Then $n+1 = size(gen(n+1)) \le \varphi_{gen}(size(n+1)) = \varphi_{gen}(1) = n$. This contradiction shows that *gen* is not definable. □

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 4.4.

Theorem 6.9 *Under the translations of Theorem 4.4, we have the following equivalences of languages:*

- $\mathcal{NRL}^{\rm nat}(powerset, gen) \simeq \mathcal{BQL}(powerbag)$.
- $\mathcal{NRL}^{\rm nat}(s\_sri, gen) \simeq \mathcal{BQL}(b\_sri)$.

Proof. Since one inclusion was proved in Theorem 6.8, we have to prove the reverse inclusions. We do it for the power operators; the other equivalence is similar. Let t be a type that may involve bags but not sets, and let $s = to\_set(t)$; see the proof of Theorem 4.4. We have to find a function f in $\mathcal{NRL}^{\rm nat}(powerset, gen)$ such that for any bag B of type $\{\!|t|\!\}$ the following holds:

$$to\_set_{\{\!|\{\!|t|\!\}|\!\}} \circ powerbag(B) \;=\; f \circ to\_set_{\{\!|t|\!\}}(B)$$

Elements of the set $X = \mathit{to\_set}_{\{\!|t|\!\}}(B)$ are pairs of type $s \times \mathbb{N}$ where the integer component indicates the number of occurrences of the element in B. We have to construct a set of type $\{\{s \times \mathbb{N}\} \times \mathbb{N}\}$ that represents $\mathit{powerbag}(B)$ under the *to_set* translation. We do this in four steps.

Step 1. For each pair (x, n) in X we create n copies of x with distinct labels of type $\mathbb{N}$.

Step 2. Powerset of the X' set created in Step 1 is taken; the result is denoted by $\mathcal{X}$.

For Step 3, we define a new equivalence test for the elements of $\{s \times \mathbb{N}\}$. Two such sets Y and Z are equivalent if $\xi(Y)$ and $\xi(Z)$ represent the same bag, where $\xi(Y) = \{(y, n) \mid (y, i) \in Y, n = \mathit{card}(\{(z, j) \in Y \mid z = y\})\}$. That is, Y and Z are equivalent iff $\xi(Y) = \xi(Z)$. Now,

Step 3. Each set in $\mathcal{X}$ is paired it with the number of equivalent sets in $\mathcal{X}$, thus creating the object of type $\{\{s \times \mathbb{N}\} \times \mathbb{N}\}$.

Step 4. The function ξ is applied to the first component of every element of the outcome of Step 3.

It can be seen that the outcome of Step 4 is $\mathit{to\_set}_{\{\!|\{\!|t|\!\}|\!\}} \circ \mathit{powerbag}(B)$. Indeed, by pairing each element of type s with n distinct values, where n is its multiplicity, we simulate *powerbag* correctly, and then Steps 3 and 4 put the result in the right form, pairing elements with their multiplicities rather than those distinct labels.

To see that Steps 1 to 4 can be implemented in $\mathcal{NRL}^{\mathrm{nat}}(\mathit{powerset}, \mathit{gen})$, it is enough to show that Step 1 can be done and ξ can be expressed. The other operations are straightforward. The use of the *gen* primitive in Step 1 is crucial:

$$X' = \{(x, i) \mid (x, n) \in X,\ i \in \mathit{gen}_0(n)\}$$

where $\mathit{gen}_0(n) = \mathit{difference}(\mathit{gen}(n), s\_\eta(0)) = \{1, \ldots, n\}$. That ξ is definable follows from its definition and the fact that *card* is definable as $\sum \lambda x.1$. This finishes the proof. □

7 Conclusion and Future 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. It is shown that bag semantics corresponds naturally to adding aggregate functions to relational languages.

We have proved the conservative extension property of the relational counterparts of the basic bag language, and shown that it is a very powerful technique in analyzing their expressive power. We have presented a new technique for proving a number of inexpressibility results in a uniform way and showed that it works for our languages when aggregate functions are not present. When bags or aggregates are present, we proved a finite-cofiniteness property of some graph queries. This property ensures that some simple recursive queries remain inexpressible in the basic bag language, thus solving a number of open problems on the expressive power of bag languages.

Finally, the relationship between structural recursion and 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.

These results complement earlier ones [8, 9, 34, 50, etc.] obtained for relational languages. All these papers taken together are a foundation for programming with collection types using the paradigm of designing languages around operations naturally associated with their datatypes.

Future work. There are many further problems which we would like to investigate. First, we would like to know if the following is true.

Conjecture 4 $\mathcal{NRL}^{\text{aggr}}$ *has the bounded degree property.*

Answering the following questions may shed some light on this conjecture.

1. What is a logic that captures (the first-order fragment of) $\mathcal{NRL}^{\text{aggr}}$?
2. Which logics have the bounded degree property? Observe that we used only a part of Gaifman's result to prove the bounded degree property for the first-order logic. Hence we believe there is a chance to find its generalizations for other logics.

It is known that the presence of a linear order adds tremendous power to first-order query languages [3]. Our language for nested sets/bags has enough power to express a linear order at all types. It is a good framework for investigating the impact of linear orders on nested collections. However, the inexpressibility results proved for $\mathcal{BQL}$ and $\mathcal{NRL}^{\text{aggr}}$ assume an unordered type, and we do not know whether they continue to hold for ordered base types. Also, adding *gen* to $\mathcal{NRL}^{\text{aggr}}$ destroys the bounded degree property, but we conjecture that queries such as *chain* and *bbtree* remain inexpressible in $\mathcal{NRL}^{\text{aggr}}(gen)$. We do not know of any techniques that would allow us to answer these questions.

It would be interesting to extend optimizations for set languages given by equational theory of the corresponding monad [8, 59, 60] to bags and, in particular, to $\mathcal{BQL}$ queries. It is known [14] that many optimizations that work well for sets do not carry over to bags. Furthermore, it was shown recently [20] that equational theories of bag languages with monus are rather complicated. Nevertheless, we believe some useful optimizations can still be found.

Finally, we would like to use the results of this report as a basis for extending the approach of Buneman et al. [10] and Libkin [31, 32] to study bags with partial information and to program with them. Initial results in this direction are reported in [38].

List of the langauages

Language	Definition in
$\mathcal{NRL}$ and its components: $\mathcal{NRA}, \mathcal{NRC}$ and $\mathcal{RSA}$	Section 2
$\mathcal{NRL}(eq)$	Section 2
$\mathcal{NBL}$ and its components: $\mathcal{NBA}, \mathcal{NBC}$ and $\mathcal{RBA}$	Subsection 3.1
$\mathcal{BQL}$	Subsection 3.2
$\mathcal{NRL}^{\mathrm{nat}}$	Subsection 4.2
$\mathcal{NRL}^{\mathrm{aggr}}$	Subsection 4.3
$\mathcal{BQL}(powerbag)$	Subsection 6.1
$\mathcal{BQL}(b\_sri)$	Subsection 6.1
$\mathcal{BQL}(loop)$	Subsection 6.2
$\mathcal{NRL}^{\mathrm{nat}}(powerset), \mathcal{NRL}^{\mathrm{nat}}(s\_sri)$	Subsection 6.4
$\mathcal{NRL}^{\mathrm{nat}}(powerset, gen), \mathcal{NRL}^{\mathrm{nat}}(s\_sri, gen)$	Subsection 6.4

Acknowledgements. Peter Buneman gave us the initial inspiration and provided many helpful suggestions. We are grateful to Stéphane Grumbach for numerous comments and suggestions. We would like to thank all who made helpful comments on earlier versions of this paper: Kousha Etessami, Jean Gallier, Neil Immerman, Paris Kanellakis, Jan Paredaens, Dan Suciu, Val Tannen, Jan Van den Bussche, Bennet Vance, and Scott Weinstein. Finally, we thank Latha Colby for a careful reading of the final version of this paper.

Most of this work was done when both authors were at the University of Pennsylvania. We gratefully acknowledge the support of an AT&T Doctoral Fellowship and NSF Grant IRI-90-04137 (for Libkin) and NSF Grant IRI-90-04137 and ARO Grant DAALO3-89-C-0031-PRIME (for Wong).

References

[1] S. Abiteboul, C. 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.

[2] S. Abiteboul and G. Hillebrand, Space usage in functional query languages, *in* "LNCS 893: Proceedings of 5th International Conference on Database Theory," Springer Verlag, January 1995.

[3] S. Abiteboul, R. Hull, V. Vianu, "Foundations of Databases," Addison Wesley, 1994.

[4] J. Albert, Algebraic properties of bag data types, *in* "Proceedings of 17th International Conference on Very Large Data Bases," 1991.

[5] R.B. Boppana, M. Sipser, The Complexity of Finite Functions, *in* "Handbook of Theoretical Computer Science," Volume A, Chapter 14, pages 759–804, North Holland, 1990.

[6] V. Breazu-Tannen, P. Buneman, S. Naqvi, Structural recursion as a query language, *in* "Proceedings of 3rd International Workshop on Database Programming Languages," Naphlion, Greece, August 1991.

[7] V. Breazu-Tannen, 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.

[8] V. Breazu-Tannen, P. Buneman, L. Wong, Naturally embedded query languages, *in* "LNCS 646: Proceedings of International Conference on Database Theory," Berlin, Germany, October 1992. Full paper to appear in *Theoretical Computer Science*.

[9] P. Buneman, L. Libkin, D. Suciu, V. Tannen, L. Wong, Comprehension syntax, *SIGMOD Record* **23** (1994), 87–96.

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

[11] L. Cardelli, Types for data-oriented languages, *in* "LNCS 303: Proceedings of International Conference on Extending Database Technology," 1988.

[12] A. Chandra, D. Harel, Structure and complexity of relational queries, *Journal of Computer and System Sciences* **25** (1982), 99–128.

[13] L. S. Colby, A recursive algebra for nested relations, *Information Systems* **15**, No. 5 (1990), 567–582.

[14] S. Chaudhuri, M. Y. Vardi, Optimization of *real* conjunctive queries, *in* "Proceedings of 12th ACM Symposium on Principles of Database Systems," Washington, D. C., May 1993.

[15] M.P. Consens, A.O. Mendelzon, Low complexity aggregation in GraphLog and Datalog, *Theoretical Computer Science* **116**, No. 1 (1993), 95–116.

[16] K. Etessami, Counting quantifiers, successor relations, and logarithmic space, *in* "Proceedings of 10th IEEE Conference on Structure in Complexity Theory," May 1995.

[17] R. Fagin, Finite model theory — a personal perspective, *Theoretical Computer Science* **116**, No. 1 (1993), 3–32.

[18] R. Fagin, L. Stockmeyer, M. Vardi, On monadic NP vs monadic co-NP, *in* "Proceedings of 8th IEEE Conference on Structure in Complexity Theory," May 1993.

[19] H. Gaifman, On local and non-local properties, *in* "Proceedings of the Herbrand Symposium, Logic Colloquium '81," North Holland, 1982.

[20] T. Griffin, L. Libkin, "Incremental maintenance of views with duplicates," *in* "Proceedings of ACM-SIGMOD International Conference on Management of Data," San Jose, CA, May 1995.

[21] S. Grumbach, T. Milo, Towards tractable algebras for bags, *in* "Proceedings of 12th ACM Symposium on Principles of Database Systems," Washington, D. C., May 1993. Full version to appear in JCSS.

[22] M. Gyssens, D. Van Gucht, The powerset algebra as a natural tool to handle nested database relations, *Journal of Computer and System Sciences* **45** (1992), 76–103.

[23] R. Hull, J. Su, Domain independence and the relational calculus. *Acta Informatica* **31** (1994), 513–524.

[24] N. Immerman, Languages that capture complexity classes, *SIAM Journal of Computing* **16** (1987), 760–778.

[25] N. Immerman, S. Patnaik, D. Stemple, The expressiveness of a family of finite set languages, *in* "Proceedings of 10th ACM Symposium on Principles of Database Systems," May 1991.

[26] L. A. Jategaonkar, J. C. Mitchell, ML with extended pattern matching and subtypes, *in* "Proceedings of ACM Conference on LISP and Functional Programming," Snowbird, Utah, July 1988.

[27] D. Johnson, A catalog of complexity classes, *in* "Handbook of Theoretical Computer Science," Volume A, Chapter 2, pages 67–161, North Holland, 1990.

[28] P. Kanellakis, Elements of relational database theory, *in* "Handbook of Theoretical Computer Science," Volume B, Chapter 17, pages 1075-1176, North Holland, 1989.

[29] A. Klausner, N. Goodman, Multirelations: Semantics and languages, *in* "Proceedings of 11th International Conference on Very Large Data Bases," Stockholm, August 1985.

[30] A. Klug, Equivalence of relational algebra and relational calculus query languages having aggregate functions, *Journal of the ACM* **29**, No. 3 (1982), 699–717.

[31] L. Libkin, A relational algebra for complex objects based on partial information, *in* "LNCS 495: Mathematical Fundamentals of Database Systems," Springer Verlag, May 1991.

[32] L. Libkin, Approximation in databases, *in* "LNCS 893: Proceedings of 5th International Conference on Database Theory," Springer Verlag, January 1995.

[33] L. Libkin, Normalizing incomplete databases, *in* "Proceedings of 14th ACM Symposium on Principles of Database Systems," San Jose, CA, May 1995.

[34] L. Libkin, L. Wong, Semantic representations and query languages for or-sets, *in* "Proceedings of 12th ACM Symposium on Principles of Database Systems," Washington, D. C., May 1993. Full version to appear in JCSS.

[35] L. Libkin, L. Wong, Some properties of query languages for bags, *in* "Proceedings of 4th International Workshop on Database Programming Languages," Manhattan, New York, August 1993.

[36] L. Libkin, L. Wong, Aggregate functions, conservative extension, and linear orders, *in* "Proceedings of 4th International Workshop on Database Programming Languages," Manhattan, New York, August 1993.

[37] L. Libkin, L. Wong, Conservativity of nested relational calculi with internal generic functions, *Information Processing Letters* **49** (1994), 273–280.

[38] L. Libkin, L. Wong, On representation and querying incomplete information in databases with bags, *Information Processing Letters* **56** (1995), 209–214.

[39] M. Machtey, P. Young, "An introduction to the General Theory of Algorithms," North Holland, 1978.

[40] E. MOGGI, Notions of computation and monads, *Information and Computation* **93** (1991), 55–92.

[41] P. ODIFREDDI, "Classical Recursion Theory," North Holland, 1989.

[42] A. OHORI, P. BUNEMAN, V. BREAZU-TANNEN, Database programming in Machiavelli, a polymorphic language with static type inference, *in* "Proceedings of ACM-SIGMOD International Conference on Management of Data," Portland, Oregon, June 1989.

[43] G. OZSOYOGLU, Z. M. OZSOYOGLU, V. MATOS, Extending relational algebra and relational calculus with set-valued attributes and aggregate functions, *ACM Transactions on Database Systems* **12**, No. 4 (1987), 566–592.

[44] J. PAREDAENS, Private communication, Collection Types Workshop, Bellcore, February 1993.

[45] J. PAREDAENS, P. DE BRA, M. GYSSENS, D. VAN GUCHT, "The Structure of the Relational Data Model," Springer-Verlag, Berlin, 1989.

[46] J. PAREDAENS, D. VAN GUCHT, Converting nested relational algebra expressions into flat algebra expressions, *ACM Transaction on Database Systems* **17**, No. 1 (1992), 65–93.

[47] H. E. ROSE, "Subrecursion: Functions and Hierarchies," Clarendon Press, Oxford, 1984.

[48] Y. SARAIYA, "Fixpoints and optimizations in a language based on structural recursion on sets," Manuscript, December 1992.

[49] H.-J. SCHEK, M. H. SCHOLL, The relational model with relation-valued attributes, *Information Systems* **11**, No. 2 (1986), 137–147.

[50] D. SUCIU, Fixpoints and bounded fixpoints for complex objects, *in* "Proceedings of 4th International Workshop on Database Programming Languages," Manhattan, New York, August 1993.

[51] D. SUCIU, V. TANNEN, A query language for NC, *in* "Proceedings of 13th ACM Symposium on Principles of Database Systems," Minneapolis, Minnesota, May 1994.

[52] D. SUCIU, J. PAREDAENS, Any algorithm in the complex object algebra with powerset needs exponential space to compute transitive closure, *in* "Proceedings of 13th ACM Symposium on Principles of Database Systems," Minneapolis, May 1994.

[53] V. TANNEN, Tutorial: Languages for collection types, *in* "Proceedings of 13th Symposium on Principles of Database Systems," Minneapolis, May 1994.

[54] S. J. THOMAS, P. C. FISCHER, Nested relational structures, *in* "Advances in Computing Research: The Theory of Databases," JAI Press, 1986.

[55] J. VAN DEN BUSSCHE, "Complex object manipulation through identifiers: An algebraic perspective," Technical Report 92-41, University of Antwerp, Department of Mathematics and Computer Science, Universiteitsplein 1, B-2610 Antwerp, Belgium, September 1992.

[56] J. VAN DEN BUSSCHE, J. PAREDAENS, "The expressive power of structured values in pure OODB," *in* "Proceedings of 10th ACM Symposium on Principles of Database Systems," Denver, Colorado, May 1991.

[57] J. D. ULLMAN, "Principles of Database and Knowledgebase Systems," Volume I, Computer Science Press, 1989.

[58] P. WADLER, Comprehending monads, *Mathematical Structures in Computer Science* **2** (1992), 461–493.

[59] L. WONG, Normal forms and conservative properties for query languages over collection types, *in* "Proceedings of 12th ACM Symposium on Principles of Database Systems," Washington D. C., May 1993.

[60] L. WONG, "Querying Nested Collections," PhD Thesis, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, PA 19104, August 1994.