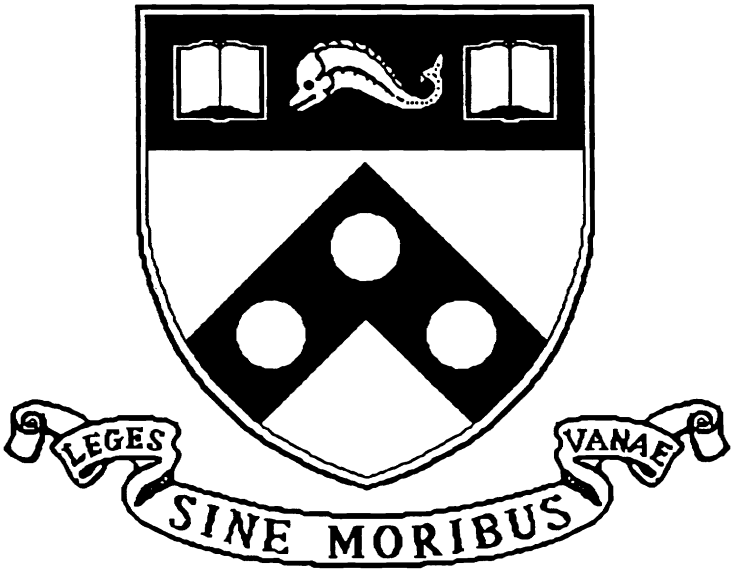
Query Languages for Bags

MS-CIS-93-36
LOGIC & COMPUTATION 59

Leonid Libkin
Limsoon Wong



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

March 1993

Query Languages for Bags

Leonid Libkin* Limsoon Wong†

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

Abstract

In this paper we study theoretical foundations for programming with bags. We fully determine the strength of many polynomial bag operators relative to an ambient query language. Then picking the strongest combination of these operators we obtain the yardstick nested bag query language $\mathcal{NBL}(monus, unique)$. The relationship between nested relational algebra and various fragments of $\mathcal{NBL}(monus, unique)$ is investigated. The precise amount of extra power that $\mathcal{NBL}(monus, unique)$ possesses over the nested relational algebra is determined. An ordering for dealing with partial information in bags is proposed and a technique for lifting a linear order at base types to linear order at all types is presented. This linear order is used to prove the conservative extension property for several bag languages. Using this property, we prove some inexpressibility results for $\mathcal{NBL}(monus, unique)$. In particular, it can not test for a property that is simultaneously infinite and co-infinite (for example, parity). Then non-polynomial primitives such as powerbag, structural recursion and bounded loop are studied. Structural recursion on bags is shown to be strictly more powerful than the powerbag primitive and it is equivalent to the bounded loop operator. Finally, we show that the numerical functions expressible in $\mathcal{NBL}(monus, unique)$ augmented by structural recursion are precisely the primitive recursive functions.

1 Introduction

Sets and bags are closely related data structures. While sets have been studied intensively by the theoretical database community, bags have not received the same amount of attention. However, real implementations frequently use bags as the underlying data model. For example, the "select distinct" construct and the "select average of column" construct of SQL can be better explained if bags instead of sets are used. In this report, query languages for bags are examined with two objectives in mind. The first objective is to suggest a good candidate for a bag query language yardstick. Towards this end, the relative expressive power of various query languages for bags is investigated; the expressive power of these bag languages relative to the nested relational query language of Breazu-Tannen, Buneman, and Wong [5] is studied; and an inquiry into certain fundamental questions on the expressive power of bags is made. The second objective is to use

*Supported in part by NSF Grant IRI-90-04137 and AT&T Doctoral Fellowship.
†Supported in part by NSF Grant IRI-90-04137 and ARO Grant DAALO3-89-C-0031-PRIME.

insights gained from bags to enhance nested relational query languages. To achieve this goal, various ways of augmenting the language of [5] to gain the expressive power of the yardstick bag language are considered.

In an earlier paper [5], Breazu-Tannen, Buneman, and Wong studied the use of monad [24] and structural recursion [3] for querying sets. We use this language as our ambient set language. In this report, the same syntax is given a semantics based on bags in section 2. We use this language as our ambient bag language. This highlights the *uniform* manipulation of sets and bags using monad as noted by Wadler [36] and structural recursion as noted by Breazu-Tannen and Subrahmanyam [4]. Incidentally, the equivalence between nested relational algebra and nested relational calculus in [5] carries over here effortlessly as an equivalence between nested bag algebra and nested bag calculus.

The ambient bag language is inadequate in expressive power as it stands. In section 3, additional primitives are proposed and their relative strength with respect to the ambient language is fully investigated. The primitive *unique* which eliminates duplicates from a bag is shown to be independent of the other primitives. A similar result was obtained by Van den Bussche and Paredaens in the setting of pure object oriented databases [11]. The primitive *monus* which subtracts one bag from another is proved to be the strongest amongst the remaining primitives. This result was independently obtained by Albert [2]. However, his investigation on relative strength is not as complete as this report.

The relationship of bag and set queries is studied in Section 4. It is shown that the class of set functions computed by the ambient bag language endowed with equality on base types, test for emptiness, and *unique*, is precisely the class of functions computed by the nested relational language of [5]. Furthermore, if equality at all types is available, then the former strictly includes the latter. The importance of *unique* is also demonstrated in this section by showing that there is a function expressible by [5] that is inexpressible by the yardstick bag language if *unique* is removed. Grumbach and Milo also examined the relationship between sets and bags [12]. However, they considered languages which have powerset and powerbag operators as primitives. These operators are impractical because they have exponential data complexity and they are too coarse grain. Also, Grumbach and Milo considered set functions on relations whose height of set nesting is at most 2. No such limit is imposed in this report. Moreover, the languages considered in the main parts of this report all have polynomial complexity.

The relationship between sets and bags can be examined from a different perspective. In the remainder of section 4, we investigate augmenting the set language of [5] to endow it with precisely the expressive power of our yardstick bag language. This is achieved by adding natural numbers, multiplication, subtraction, and a summation construct to the nested relational language. This also illustrates the natural relationship between bags and numbers. This relationship is further exploited in section 5 when rational division is added to the set language as well. The resulting nested relational language has the ability to express queries such as "select average from column" and "select count from column." In a couple of early work on this topic, Klausner and Goodman proposed a notion of hiding as an explanation of the semantics of aggregate operators [18] and Klug defined a collection of aggregate operators such as $average_1$, $average_2$, ..., one for each column of a flat relation [19]. Ozsoyoglu, Ozsoyoglu, and Matos extended Klug's approach to nested relations [27]. The augmented nested relational language is a natural and elegant generalization of these three proposals.

Orderings on bags are necessary to deal with such problems as partial information or effective storage strategies. We study several ways to order bags in section 5. First, the approach of Libkin and Wong [22] is extended to bags and the resulting order is shown to be tractable. Second, we present a way to lift linear orders at base types to linear orders at arbitrary types. It is easily implemented in a simple extension of the ambient language. This linear order is at the heart of proving conservative extension properties for various languages studied in this paper.

Queries expressible in the augmented language are proved, in section 6, to be independent of the height of set nesting of intermediate results. This is a significant generalization of the conservative extension result of Wong [38] and Paredaens and Van Gucht [29]. In particular, it implies that nested relational queries whose input and output are flat relations can be expressed in a language like SQL, even if aggregate operators such as average and count are used. The conservativeness of transitive closure, bounded fixpoints, and powerset operators is obtained as a remarkable corollary.

This result is then used in section 7 to prove several fundamental properties of bag languages. In particular, the inexpressibility of properties (such as parity test) on natural numbers that are simultaneously infinite and co-infinite. Another consequence is that subbag test is inexpressible using just *unique* and equality tests.

Breazu-Tannen, Buneman, and Wong proved that the power of structural recursion on sets can be obtained by adding a powerset operator to their language [5]. However, this result is contingent upon the restriction that every type has a finite domain. In section 8, the powerbag primitive of Grumbach and Milo [12] is contrasted with structural recursion on bags. In particular, the latter is shown to be strictly more expressive than the former. As mentioned earlier, although a powerbag primitive increases expressive power considerably, it is difficult to express algorithms that are efficient. While structural recursion does not have this deficiency, it requires the satisfaction of certain preconditions that cannot be automatically verified [4]. In section 8, a bounded loop construct which does not require the verification of any precondition is introduced. It is shown to be equivalent in expressive power to structural recursion over sets, bags, as well as lists. This confirms the intuition that structural recursion is just a special case of bounded loop. Furthermore, in contrast to the powerbag primitive which gives us all elementary functions [12], structural recursion gives us all primitive recursive functions.

2 The ambient query language

We first present the nested relational language of Breazu-Tannen, Buneman, and Wong [5]. Then we describe the ambient bag language obtained from it.

2.1 The nested relational language

The nested relational language proposed by Breazu-Tannen, Buneman, Wong [5] is denoted by $\mathcal{NRL}$ here. It has three equally expressive components that can be freely combined: the nested relational algebra $\mathcal{NRA}$, the nested relational calculus $\mathcal{NRC}$, and relative set abstraction $\mathcal{RSA}$.

Types. The types of $\mathcal{NRL}$ are complex object types s and function types $s \to t$ where s and t are complex object types. Complex object types are given by

$$s ::= \mathit{unit} \mid b \mid s \times s \mid \{s\}$$

where *unit* is a special base type containing exactly the distinguished value denoted by $()$, b ranges over an unspecified collection of base types, $s \times t$ are tuples whose first component is of type s and second component is of type t, and $\{s\}$ are finite sets whose elements are of type s.

Expressions (sometimes called *morphisms)* of $\mathcal{NRA}$, $\mathcal{NRC}$ and $\mathcal{RSA}$ are constructed using the rules in figure 2.1 The type superscripts are omitted in subsequent sections as they can be inferred (see [17, 26] for example). The semantics of these constructs has been fully explained in [5]. We briefly repeat their semantics here.

EXPRESSIONS OF $\mathcal{NRA}$

Category with Products

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

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

Set Monad

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

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

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 Monad

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

EXPRESSIONS OF $\mathcal{RSA}$

All expressions of $\mathcal{NRC}$ without $\bigcup\{e_1 \mid x \in e_2\}$ and

Relative set abstraction construct

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

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

- Kc is the constant function that produces the constant c.
- id is the identity function.
- $g \circ h$ is the composition of functions g and h; that is, $(g \circ h)(d) = g(h(d))$.
- The bang ! produces $()$ on all inputs. π_1 and π_2 are the two projections on pairs.
- $\langle g, h \rangle$ is pair formation; that is, $\langle g, h \rangle(d) = (g(d), h(d))$.
- $K\{\}$ produces the empty 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 $\{1,2,3,5,7,4\}$.
- $s\_map(f)$ applies f to every item in the input set; for example $s\_map(\lambda x.1+x)\{1,2,3\}$ yields $\{2,3,4\}$.
- $s\_\rho_2(x,y)$ 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 $s\_\mu \circ s\_map(\lambda x.e_1)$.
- $\{e \mid x_1 \in e_1, ..., x_n \in e_n\}$ is equivalent to $\bigcup\{\ldots\bigcup\{\{e\} \mid x_n \in e_n\}\ldots \mid x_1 \in e_1\}$.

The whole of $\mathcal{NRL}$ is used in many places of this report. However, in many of our 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 [5, 38].

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.* □

In [5], booleans are represented by $\{()\}$ (truth) and $\{\}$ (falsity), the two values of type $\{unit\}$. It was shown that after adding for each complex object type s, an equality test primitive $eq^s : s \times s \to \{unit\}$, $\mathcal{NRL}$ expresses all nested relational operations of the well-known algebra of Thomas and Fischer [33]. In fact, this result can be strengthened because the converse is also true if a few constant relations are added to the algebra of Thomas and Fischer (which is known to be equivalent to the language to Colby [9] and to the language of Schek and Scholl [31]). Also, 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 $\mathcal{NRL}(eq) \simeq \mathcal{NRL}(=, bool, cond) \simeq Thomas\&Fischer \simeq Schek\&Scholl \simeq Colby.$ □

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 $\lambda X.\{(\pi_1\ x,\ \{\pi_2\ y \mid y \in X, w \in (\pi_1\ y\ eq\ \pi_1\ x)\}) \mid x \in X\}$.

2.2 The nested bag language

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 $\mathcal{RBA}$. Following Wadler [36] and Watt and Trinder [37], the bag languages are obtained by replacing the set monad constructs in the nested relational languages by the corresponding bag monad constructs. This yields a uniform method for manipulating collection types such as sets and bags. We list only the parts that are changed.

Types. $\mathcal{NBL}$ has the same types as $\mathcal{NRL}$ but uses bags instead of sets. That is,

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

where $\{\!|s|\!\}$ 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.

Expressions. The expressions of $\mathcal{NBL}$ are given in figure 2.2.

The semantics of these constructs is similar to the semantics of $\mathcal{NRL}$ except duplicates are not eliminated. 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, $b\_map(\lambda x.1+x)\ \{\!|1,2,1,6|\!\}$ evaluates to $\{\!|2,3,2,7|\!\}$. $K\{\!|\}$ 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 flatmap 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 $\biguplus\{\!|\ldots\biguplus\{\!|\{\!|e|\!\} \mid x_n \in e_n|\!\}\ldots| x_1 \in e_1|\!\}$. 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 $\mathcal{NRL}$ [5].

Proposition 2.3 *$\mathcal{NBA}$, $\mathcal{NBC}$, 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 component that is most convenient.

3 Relative strength of bag operators

As mentioned earlier, the presence of equality tests elevates $\mathcal{NRL}$ from a language that merely has structural manipulation capability to a full fledge 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 yardstick, 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 [2] and Grumbach and Milo [12]. In contrast to Grumbach and Milo [12] who included a powerbag operator as a primitive, all operators

EXPRESSIONS OF $\mathcal{NRA}$

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

Bag Monad

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

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

EXPRESSIONS OF $\mathcal{NBC}$

$\lambda-$calculus with products as in $\mathcal{NRC}$

Bag Monad

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

EXPRESSIONS of $\mathcal{RBA}$

All operations of $\mathcal{NBC}$ without $\biguplus \{\!|e_1 \mid x^s \in e_2|\!\}$ 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 $\mathcal{NBL}$

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. A bag is just an *unordered* collection of items. $count(d, B)$ is defined to be the number of times the object d occurs in the bag B. The bag operations to be considered are listed below.

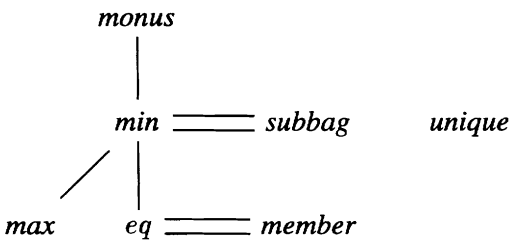
- *monus* : $\{|s|\} \times \{|s|\} \to \{|s|\}$. $monus(B_1, B_2)$ evaluates to a B such that for every $d : s$, $count(d, B) = count(d, B_1) - count(d, B_2)$ if $count(d, B_1) > count(d, B_2)$; and $count(d, B) = 0$ otherwise.
- *max* : $\{|s|\} \times \{|s|\} \to \{|s|\}$. $max(B_1, B_2)$ evaluates to a B such that for every $d : s$, $count(d, B) = \max(count(d, B_1), count(d, B_2))$.
- *min* : $\{|s|\} \times \{|s|\} \to \{|s|\}$. $min(B_1, B_2)$ evaluates to a B such that for every $d : s$, $count(d, B) = \min(count(d, B_1), count(d, B_2))$.
- $eq : s \times s \to \{|unit|\}$. $eq(d_1, d_2) = \{|()|\}$ if $d_1 = d_2$; it evaluates to $\{||\}$ otherwise. That is, we are simulating booleans a bag of type $\{|unit|\}$. True is represented by the singleton bag $\{|()|\}$ and False is represented by the empty bag $\{||\}$.
- *member* : $s \times \{|s|\} \to \{|unit|\}$. $member(d, B) = \{|()|\}$ if $count(d, B) > 0$; it evaluates to $\{||\}$ otherwise.
- *subbag* : $\{|s|\} \times \{|s|\} \to \{|unit|\}$. $subbag(B_1, B_2) = \{|()|\}$ if for every $d : s$, $count(d, B_1) \leq count(d, B_2)$; it evaluates to $\{||\}$ otherwise.
- *unique* : $\{|s|\} \to \{|s|\}$. $unique(B)$ eliminates duplicates from B. That is, for every $d : s$, $count(d, B) > 0$ if and only if $count(d, unique(B)) = 1$.

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

Proposition 3.1 *Every function definable in $\mathcal{NBL}(monus, max, min, eq, member, unique)$ has polynomial time and space complexity with respect to the size of input.* □

In the remainder of this section, the expressive power of these primitives is compared. The result of comparisons is a complete characterization of their relative expressive power: *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*. As a consequence of these results, $\mathcal{NBL}(monus, unique)$ can be considered as the most powerful candidate as a standard bag query language. These results are summarized by the following

Theorem 3.2



Let us first prove the easy expressibility results. After that, the harder inexpressibility results are presented.

Proposition 3.3 *1. max can be expressed in $\mathcal{NBL}(monus)$*

2. *min can be expressed in $\mathcal{NBL}(monus)$*
3. *eq can be expressed in $\mathcal{NBL}(monus)$*
4. *subbag can be expressed in $\mathcal{NBL}(monus)$*
5. *subbag can be expressed in $\mathcal{NBL}(eq, max)$*
6. *member can be expressed in $\mathcal{NBL}(eq)$*
7. *eq can be expressed in $\mathcal{NBL}(member)$*
8. *eq can be expressed in $\mathcal{NBL}(min)$*
9. *subbag can be expressed in $\mathcal{NBL}(min)$*
10. *min can be expressed in $\mathcal{NBL}(subbag)$*
11. *max can be expressed in $\mathcal{NBL}(min)$*

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

1. B_1 *max* $B_2 := B_2 \uplus (B_1$ *monus* $B_2)$
2. B_1 *min* $B_2 := B_1 monus (B_1 monus B_2)$
3. d_1 *eq* $d_2 := \{\!|()|\!\}$ *monus* $(R_{12} \uplus R_{21})$ where R_{ij} is $\bigcup \{\!|\{\!|()|\!\}$ | $x \in \{\!|d_i|\!\}$ *monus* $\{\!|d_j|\!\}\!\}$.
4. B_1 *subbag* $B_2 := B_1$ *eq* $(B_1$ *min* $B_2)$
5. B_1 *subbag* $B_2 := B_2$ *eq* $(B_1$ *max* $B_2)$
6. d *member* $B := (\{\!|() \mid x \in B, y \in (x$ *eq* $d)|\!\}$ *eq* $\{\!|\}\!|)$ *eq* $\{\!|\}\!|$
7. d_1 *eq* $d_2 := d_1$ *member* $\{\!|d_2|\!\}$
8. d_1 *eq* $d_2 := \{\!|() \mid x \in \{\!|d_1|\!\}$ *min* $\{\!|d_2|\!\}\!|\!\}$.
9. B_1 *subbag* $B_2 := B_1$ *eq* $(B_1$ *min* $B_2)$
10. B_1 *min* $B_2 := E \uplus F_{12} \uplus F_{21}$ where E is B_1 *intersection* B_2, and F_{ij} is $\{\!|x \mid x \in B_i$ *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.
11. B_1 *max* $B_2 := E \uplus F_{12} \uplus F_{21}$ where E is B_1 *intersection* B_2 and F_{ij} is $\{\!|x \mid x \in B_j$ *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 all nonmonotonic primitives are interdefinable [5], 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 3.4 *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: $d_1 \sqsubseteq_b d_2$; $(d_1, d_2) \sqsubseteq_{s \times t} (d_1', d_2')$ 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 3.5 *unique cannot be expressed in* $\mathcal{NBL}(monus)$.

Proof. The technique of Wong [38] can be readily adapted to show that the rewriting system below is strongly normalizing.

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

It is not difficult to show that the rewriting system obtained by adjoining the rule below to the above system is weakly normalizing:

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

Now we argue that no normal form under these rewrite rules implement *unique*. Suppose $\lambda R.e$ is a normal form that expresses *unique*. Let o be a bag of k *apples* (where *apples* is a new unspecified base type). Let $select(p) : \{\!|s|\!\} \rightarrow \{\!|s|\!\}$, where $p : s \rightarrow bool$ is a predicate, be a selection function. That is, $select(p)(R)$ evaluates to a B such that for every d, if $p(d)$ then $count(d, B) = count(d, R)$ else $count(d, B) = 0$. Then the proposition follows from the claim below.

Claim. Let A be a subexpression of e of the form $\{\!|e' \mid \Delta|\!\}$ such that the only free variable in A is R and $\{\!|\pi \ldots \pi \ x \mid x \in A|\!\}$ is a bag of apples. Let p be any predicate. Then $select(p)((\lambda R.A)(o))$ evaluates to a bag of $m \cdot k$ items for some m.

Proof of claim. We proceed by induction on A. Since e is in normal form, A can have two possible forms. A can have the form $\{\!|e' \mid x \in R, \Delta'|\!\}$. This case is immediate. Alternatively, A can have the form $\{\!|e' \mid y \in B \ monus \ C|\!\}$. In this case, B and C must be constructed from $\uplus$ and *monus* of expressions D_i in the same form as A. Since selection is injective, it can be pushed inside *monus* as a new predicate $q := p \circ (\lambda y.e')$. By hypothesis, $select(q)((\lambda R.D_i)(o))$ evaluates to $m_i \cdot k$ items. Clearly no matter how the $select(q)((\lambda R.D_i)(o))$ are added or subtracted, the result is a multiple of k items. □

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

Proof. Let e be an expression of $\mathcal{NBL}(subbag)$ in normal form (induced by the rewriting 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 assigns 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 to 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 $e\theta$ evaluates 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. □

Proposition 3.7 *Define MIN : $\{\!|\{\!|s|\!\}|\!\} \rightarrow \{\!|s|\!\}$ as the function with the following semantics: $MIN(R) = B$ such that for every d, $count(b, B) = \min\{count(d, X) \mid X \in R\}$. Then*

1. *MIN cannot be expressed in $\mathcal{NBL}(monus)$*
2. *MIN cannot be expressed in $\mathcal{NBL}(unique, member)$*
3. *MIN can be expressed in $\mathcal{NBL}(unique, member, max)$*
4. *subbag cannot be expressed by $\mathcal{NBL}(member)$.*
5. *max cannot be expressed by $\mathcal{NBL}(unique, member)$.*

Proof. The last two items are immediate consequences of the first three items.

1. Since *unique* cannot be expressed in $\mathcal{NBL}(monus)$, it suffices to show that it is expressible in $\mathcal{NBL}(MIN, eq)$. Clearly, $unique(B) := MIN\,\{\!|\{\!|r|\!\} \uplus \{\!|x \mid x \in B, y \in (x\ eq\ r)\ eq\ \{\!|\}\!\} \mid r \in B|\!\}$.
2. 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 the natural numbers, some limited arithmetics, 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 7 that $\leq : \mathbb{N} \times \mathbb{N} \rightarrow bool$ 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 $\mathcal{NBL}(unique, member)$.
3. 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 3.2. The independence of *unique* was also proved by Van den Bussche and Paredaens [11] and the fact that *monus* is the strongest amongst the remaining primitives was also showed by Albert [2]. However, their comparison was incomplete. For example, the incomparability of *max* and *eq* was not reported. In contrast, the results presented in this section can be put together in theorem 3.2 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 our 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 $\mathcal{NBL}(monus, unique)$, 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. We use the following constructs for this purpose:

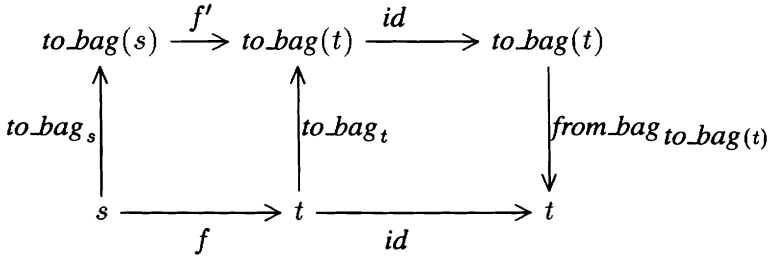
$$\frac{f : s \to t}{bs\_map(f) : \{|s|\} \to \{t\}} \qquad \frac{f : s \to t}{sb\_map(f) : \{s\} \to \{|t|\}}$$

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

Let s be a complex object type not involving bags. Then $to\_bag(s)$ is a complex object type obtained by 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.

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

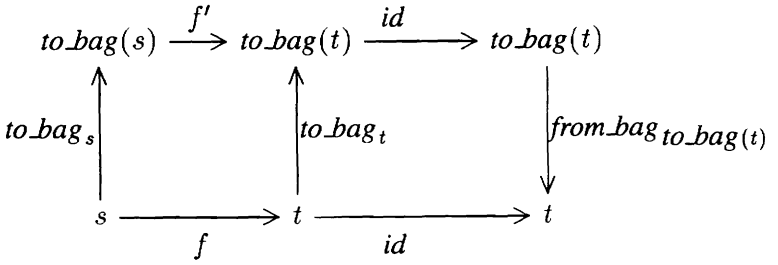
Define $\mathcal{SET}(\Gamma)$ to be the class of functions $f : s \to t$ where s and t are complex object types not involving bags and Γ is a list of primitives such that there is $f' : to\_bag(s) \to to\_bag(t)$ definable in $\mathcal{NBL}(\Gamma)$ and the diagram below commutes.



The class $\mathcal{SET}(\Gamma)$ is precisely the class of "set theoretic" functions expressible in $\mathcal{NBL}(\Gamma)$. We compare $\mathcal{SET}(\Gamma)$ with $\mathcal{NRL}(eq)$ for various bag primitives below.

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

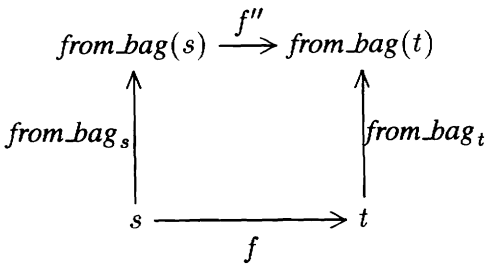
Proof. It is easy to check [38] that $\mathcal{NRL}(eq) = \mathcal{NRL}(eq_b, not)$ where $not : \{unit\} \to \{unit\}$ returns $\{()\}$ if applied to the empty set and returns $\{\}$ otherwise. Hence we prove $\mathcal{SET}(unique, eq_b, empty) = \mathcal{NRL}(eq_b, not)$ instead. To show $\mathcal{NRL}(eq_b, not) \subseteq \mathcal{SET}(unique, eq_b, empty)$, 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 $\mathcal{NBA}(eq_b, empty, unique)$ such that



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:

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

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 $\mathcal{NRA}(unique, eq_b, empty)$ there is an $f'' : from\_bag(s) \to from\_bag(t)$ in $\mathcal{NRL}(eq_b, not)$ such that



This is straightforward by defining f'' as follows:

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

□

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

Proof. Since $\mathcal{NRL}(eq)$ is a conservative extension of the flat relational algebra [38], it cannot test whether two given sets have the same cardinality. 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 $\mathcal{NRL}(eq) \subsetneqq \mathcal{SET}(monus)$ because the function which tests if two sets have equal cardinality is in the latter but not the former. The proof of $\mathcal{SET}(monus) \subsetneqq \mathcal{NRL}(eq)$ is similar to the proof of the inexpressibility of *unique* in $\mathcal{NBL}(monus)$. We present the important difference below. First a parametricity result is required.

Claim. Let b be a new uninterpreted base type. Let c_1, ..., c_k be the constants of this type. Let e be an expression of $\mathcal{NBC}(monus)$ in which these constants do not appear. Let φ be a bijection on $\{1, \ldots, k\}$. Let θ be an assignment of complex objects to free variables. Then $(e\theta)\varphi = e(\theta\varphi)$, where $(e\theta)\varphi$ means replacing every c_i in $e\theta$ by $c_{\varphi(i)}$ and $\theta\varphi$ is the assignment such that $(\theta\varphi)(x) = (x\theta)\varphi$ for each x.

Proof of claim. By structural induction on e.

Now we argue that s_μ is not in $\mathcal{SET}(monus)$. Suppose to the contrary that it is in $\mathcal{SET}(monus)$. Then there is a normal form $\lambda R.e$ (induced by the rewriting system given in the previous section) such that $to\_bag \circ s\_\mu = (\lambda R.e) \circ to\_bag$. Let $o : \{\!\{\{\!\{b\}\!\}\}\!\}$ where b is a new base type and $o = \{\!\{\{\!\{c, c_1\}\!\}, ..., \{\!\{c, c_k\}\!\}\}\!\}$ where c, c_1, ..., c_k are all distinct constants of type b.

Claim. Suppose A is a subexpression of e of the form $\{\!|e' \mid \Delta|\!\}$ such that the only free variable in A is R and $\{\!|\pi \ldots \pi\ x \mid x \in A|\!\}$ is a bag of type $\{\!|b|\!\}$. Let φ be any bijection on $\{c_1, \ldots, c_k\}$. Then $count(d, A[o/R]) = count(d\varphi, A[o/R])$ for any d.

Proof of claim. Immediate by parametricity.

The theorem is immediate from the claim below by setting d to c because $k + 1 \neq m \cdot k$ for any m.

Claim. Let A be a subexpression of e of the form $\{\!|e' \mid \Delta|\!\}$ such that the only free variable in A is R and $\{\!|\pi \ldots \pi\ x \mid x \in A|\!\}$ is a bag of type $\{\!|b|\!\}$. Let φ_1, ..., φ_m be bijections on $\{c_1, \ldots, c_k\}$ such that $d\varphi_1$, ..., $d\varphi_m$ are all distinct and $\{d\varphi_1, \ldots, d\varphi_m\} = \{d\varphi \mid \varphi$ is a bijection on $\{c_1, \ldots, c_k\}$ and $d\varphi$ is a member of $A[o/R]\}$. Then $\sum_{1 \leq i \leq m} count(d\varphi_i, A[o/R]) = c \cdot k$ for some constant c.

Proof of claim. Since A is in normal form, there are two possibilities. First, A is of the form $\{\!|e' \mid y \in B\ monus\ C|\!\}$. Then B and C are in the same form as A.

$$
\begin{array}{lll}
\bullet & \sum_{1 \leq i \leq m} count(d\varphi_i, A[o/R]) & \\
= & m \cdot count(d, A[o/R]) & \text{by previous claim.} \\
= & m \cdot count(d, B[o/R]\ monus\ C[o/R]) & \\
= & m \cdot count(d, B[o/R]) \dot{-} m \cdot count(d, C[o/R]) & \\
= & \sum_{1 \leq i \leq m} count(d\varphi_i, B[o/R]) \dot{-} \sum_{1 \leq i \leq m} count(d\varphi_i, C[o/R]) & \text{by previous claim.} \\
= & c_B \cdot k \dot{-} c_C \cdot k & \text{by hypothesis.} \\
= & c \cdot k & \text{for some number } c.
\end{array}
$$

Second, A is of the form $\{\!|e' \mid x \in R, \Delta|\!\}$. Without loss of generality, suppose A is $\{\!|e' \mid x \in R, y \in x, \Delta'|\!\}$. Let φ^i_j be the bijection that maps c_i to c_j. By parametricity, $(\{\!|e' \mid \Delta'|\!\}[o/R, c_i/y])\varphi^i_j = \{\!|e' \mid \Delta'|\!\}[o/R, c_j/y]$. The case follows. □

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.

4.2 A set language equivalent to $\mathcal{NBL}(monus, unique)$

It was shown in the previous section that $\mathcal{NBL}(monus, unique)$ is the most powerful amongst the bag languages considered so far. From the foregoing discussion, this bag language is a true extension of $\mathcal{NRL}(eq)$. In this subsection, the relationship between sets and bags is studied from a different perspective. In particular, the *precise* amount of extra power $\mathcal{NBL}(monus, unique)$ possesses over $\mathcal{NRL}(eq)$ is determined. In fact, in order to give the nested relational language the expressive power of $\mathcal{NBL}(monus, unique)$, it has to be endowed with natural numbers $\mathbf{N}$ together with multiplication, subtraction, and summation as defined below.

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

Subtraction $\dot{-}: \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ (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 $\sum g : \{s\} \to \mathbf{N}$ where $g : s \to \mathbf{N}$. The semantics is as follows. $\sum g\ \{o_1, \ldots, o_n\} = g(o_1) + \ldots + g(o_n)$. Equivalently, the construct $\Sigma\{e_2 \mid x^s \in e_1\} : \mathbf{N}$ where $e_2 : \mathbf{N}$ and $e_1 : \{s\}$ is also used and is interpreted as $(\sum(\lambda x.e_2))(e_1)$.

The rest of the section is devoted to proving

Theorem 4.4 $\mathcal{NBL}(monus, unique) \simeq \mathcal{NRL}(\mathbf{N}, \Sigma, \cdot, \dot{-}, cond,\ bool, =)$.

The proof is given in two propositions below. First, we need a slightly different kind of conversion between sets and bags. Two additional devices are used: $to\_nat : \{\!|unit|\!\} \to \mathbf{N}$ takes a bag containing n items to n and $from\_nat : \mathbf{N} \to \{\!|unit|\!\}$ does the opposite thing. Let s be a complex object type not involving sets. Then $to\_set(s)$ is the type obtained by changing all bag components $\{\!|t|\!\}$ to $\{to\_set(t) \times \mathbf{N}\}$. An object $o : s$ is converted to an object $to\_set_s(o)$ of type $to\_set(s)$. An object $o : to\_set(s)$ is converted to an object $from\_set_s(o)$ of type s. The two conversion functions are defined inductively below.

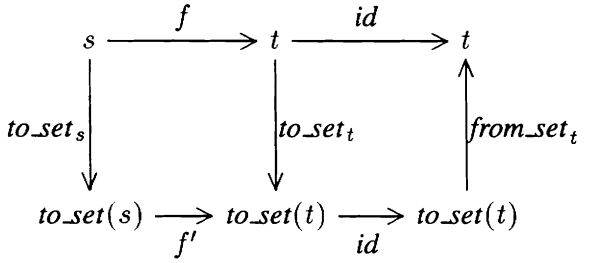
- $to\_set_{unit} := id$
- $to\_set_{s \times t} := \langle to\_set_s \circ \pi_1, to\_set_t \circ \pi_2 \rangle$
- $to\_set_{\{\!|s|\!\}} := \lambda B.bs\_map(\lambda b.(to\_set_s\ b, to\_nat\{\!|() \mid c \in B, b\ eq\ c|\!\})))\ B$
- $from\_set_{unit} := id$
- $from\_set_{s \times t} := \langle from\_set_s \circ \pi_1, from\_set_t \circ \pi_2 \rangle$

- $from\_set_{\{\!|s|\!\}} := b\_\mu \circ b\_map(b\_map\ \pi_1 \circ b\_\rho_2 \circ \langle from\_set_s \circ \pi_1, from\_nat \circ \pi_2\rangle)$

Using the above conversion, it can be shown that

Proposition 4.5 $\mathcal{NBL}(unique, monus) \subseteq \mathcal{NRL}(\mathbf{N}, \Sigma, \cdot, \dot{-}, cond,\ bool, =)$.

Proof. Since addition is definable by summation, to prove the proposition, it suffices to show that for each $f : s \to t$ in $\mathcal{NBA}(unique, monus)$, there is a $f' : to\_set(s) \to to\_set(t)$ in $\mathcal{NRL}(=, bool, cond, \mathbf{N}, +, \cdot, \dot{-}, \sum)$ such that the diagram below commutes.



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:

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

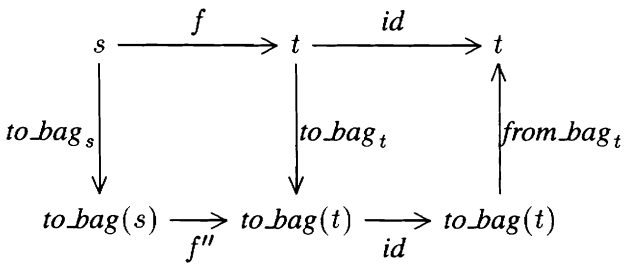
For the inclusion in the other direction, conversions similar to that in proposition 4.5 are used. Let s be a complex object type not involving sets. Then $to\_bag(s)$ is the type obtained by changing all set brackets to bag brackets and changing $\mathbf{N}$ to $\{\!|unit|\!\}$. Let o be an object of 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.

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

Then we have

Proposition 4.6 $\mathcal{NRL}(\mathbf{N}, \Sigma, \cdot, \dot{-}, cond,\ bool, =) \subseteq \mathcal{NBL}(unique, monus)$.

Proof. First note that $\mathcal{NRL}(=, bool, cond, \mathbf{N}, \cdot, \dot{-}, \sum) \simeq \mathcal{NRL}(eq, \mathbf{N}, \cdot, \dot{-}, \sum)$. Hence it suffices for us to prove that for every $f : s \to t$ in $\mathcal{NRA}(eq_b, not, \mathbf{N}, \cdot, \dot{-}, \sum)$, there is a $f'' : to\_bag(s) \to to\_bag(t)$ in $\mathcal{NBL}(monus, unique)$ such that the diagram below commutes.



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:

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

In summary, we have the following exact characterization of the relative strength between the "yardstick" bag language and the relational language of Breazu-Tannen, Buneman, and Wong: $\mathcal{NRL}(\mathbf{N}, \sum, \cdot, \dot{-}, eq) \simeq \mathcal{NBL}(unique,\ monus)$ and $\mathcal{NRL}(eq) = \mathcal{SET}(unique,\ eq_b,\ empty)$. Klug [19] and Ozsoyoglu, Ozsoyoglu, and Matos [27] had to introduce aggregate functions by repeating them for every column position of a relation. That is, $aggregate_1$ is for column one, $aggregate_2$ is for column two, etc. The $\sum$ primitive can be used to implement aggregate functions and should be seen as a generalization of their approach. Klausner and Goodman used a notion of hiding to explain the nature of aggregate functions in relational query languages [18]. In addition to projections, they introduced hiding operators that "hides" columns of a relation. Aggregate functions are then applied to the column that is left exposed. Hiding is different from projection. Let $R := \{(1,2),(1,3)\}$. Then projecting out column two on R gives $\{1\}$ while hiding column two on R gives $\{(1,[2]),(1,[3])\}$, where $[\cdot]$ signifies hidden values. The use of hiding to retain duplicates (since sets have no duplicate by definition) is a little clumsy. It is better to use bags. Since $\sum g = to\_nat \circ b\_\mu \circ sb\_map(from\_nat \circ g)$, computation on bags is perhaps a better explanation of the nature of aggregate functions.

5 Orderings on bags

The purpose of this section is twofold. First, in the spirit of [7], where databases were considered as subsets of certain partially ordered sets in order to provide rigorous mathematical treatment of partial information,

we would like to have an ordering on bags whose intuitive meaning is "being more partial". We define such on ordering using techniques proposed in [22] and give its characterization. Even though the ordering appears somewhat awkward, we demonstrate an effective algorithm to test whether two bags are comparable.

Secondly, we show that if a *linear* order is given for all base types, it can be extended to a linear order on the domain of an arbitrary type in a way that can easily be expressed in $\mathcal{NBL}(unique, monus)$. This linear order is used in section 6 to prove that various extensions of $\mathcal{NRL}(eq)$ have the conservative extension property.

As in [15, 7, 21], we assume that partiality can be expressed by means of a partial order on database objects. That is, $a \leq b$ expresses the fact that a is more partial than b or b is more informative than a. It was mentioned in [7] that many models of partial information can be captured by this very general scheme. This approach is also suitable for databases *without* partial information. In such a case, values of base types are totally unordered.

It is usually assumed that orders on the base types are given. For example, if base type is $\mathbf{N}_\perp$ whose values are natural numbers or null ($\perp$), the usual ordering is $\perp \leq n$ for any $n \in \mathbf{N}$ and any two distinct natural numbers are not comparable, see Gunter [14]. The ordering is then extended to pairs in the usual way. That is, $(x, y) \leq (x', y')$ iff $x \leq_1 x'$ and $y \leq_2 y'$. However, if one wants to extend the ordering to subsets of an ordered set, many possibilities arise. In [22] we tried to define an ordering by saying that a set X is less informative than a set Y if there is a sequence of simple updates, each leading to a more informative set. Dealing with sets, we defined the primitive updates as follows: $X \mapsto (X - \{a\}) \cup X'$ where $a \leq b$ for any $b \in X'$. Notice that if $a \notin X$, this is equivalent to augmenting X by X'.

Mathematical aspects of partial information represented by bags were studied by Vickers [35]. He represented bags over a poset as sets of pairs (a, n) where a is an element of the poset and n is the number of occurrences. Pairs were ordered in the usual way: $(a, n) \leq (b, m)$ iff $a \leq b$ and $n \leq m$. While this ordering has many nice properties, it is counterintuitive from the practical point of view. Having a bag rather than a set means that each element of a bag represents an object and if there are many occurrences of some element, then at the moment certain objects are indistinguishable. For example, initially we might have a bag of three null values, representing our knowledge about three objects. Suppose this bag $\{\!|\perp, \perp, \perp|\!\}$ is later updated to $\{\!|a, b, c|\!\}$. We want to say that the latter is more informative than the former. But that is not true in Vickers' ordering because it requires that the three nulls be replaced by three identical objects; that is, $\{\!|a, a, a|\!\}$, $\{\!|b, b, b|\!\}$, or $\{\!|c, c, c|\!\}$. Each of them is more informative than $\{\!|\perp, \perp, \perp|\!\}$ but $\{\!|a, b, c|\!\}$ is unfortunately not!

To remedy this, we follow the idea of Libkin and Wong [22] and say that a bag B_2 is more informative than a bag B_1 if B_2 can be obtained from B_1 by a sequence of updates of the following form: (1) an element a is removed from B_1 and is replaced by an element b such that b is more informative than a, or (2) an element b is added to the bag B_1. Formally, let $\langle D, \leq \rangle$ be a partially ordered set. Let $\mathcal{P}^b_{\text{fin}}(D)$ be the set of all finite bags whose elements are in D. Then, for $B_1, B_2 \in \mathcal{P}^b_{\text{fin}}(D)$, $B_1 \rightsquigarrow B_2$ iff $B_2 = (B_1 monus \{\!|a|\!\}) \uplus \{\!|b|\!\}$ where $a \leq b$ or $B_2 = B_1 \uplus \{\!|b|\!\}$. The transitive-reflexive closure of $\rightsquigarrow$ is denoted by $\trianglelefteq$. That is, we say that B_1 is *less informative* than B_2 if $B_1 \trianglelefteq B_2$.

As proved in [22], the ordering on *sets* obtained as the transitive-reflexive closure of $\mapsto$ coincides with the well-known *lower powerdomain ordering* defined as

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

A similar construction can be used to characterize $\trianglelefteq$. Let $\mathbf{N}^u$ denote the totally unordered poset whose elements are natural numbers (the superscript is used to distinguish it from $\mathbf{N}$ which in this paper denotes natural numbers with the usual ordering). For a finite bag B and an injective map $\phi : B \to \mathbf{N}^u$, which is

sometimes called *labeling*, by $\phi(B)$ we denote the set $\{(b, \phi(b)) \mid b \in B\}$. In other words, ϕ assigns a unique label to each element of a bag. If $B \in \mathcal{P}^b_{\text{fin}}(D)$, the ordering on pairs (b, n) where $b \in B$ and $n \in \mathbb{N}$" is the usual pair ordering; that is, $(b, n) \leq (b', n')$ iff $b \leq b'$ and $n = n'$.

Proposition 5.1 *The binary relation $\trianglelefteq$ on bags is a partial order. Given two bags B_1, B_2, $B_1 \trianglelefteq B_2$ iff there exist labelings ϕ and ψ on B_1 and B_2 respectively such that $\phi(B_1) \leq^\flat \psi(B_2)$.*

Proof. We write $B_1 \preccurlyeq^\flat B_2$ if there exist ϕ and ψ such that $\phi(B_1) \leq^\flat \psi(B_2)$. First demonstrate that $\preccurlyeq^\flat$ is a partial order. It is obviously reflexive.

To prove transitivity, let $B_1 \preccurlyeq^\flat B_2$ and $B_2 \preccurlyeq^\flat B_3$. That is, $\alpha(B_1) \leq^\flat \beta(B_2)$ and $\phi(B_2) \leq^\flat \psi(B_3)$. Let γ be a bijection on $\mathbb{N}$ such that $\gamma \circ \beta = \phi$. Define δ as $\gamma \circ \alpha$. Then for every $b \in B_1$ there is $b' \in B_2$ such that $b \leq b'$ and $\alpha(b) = \beta(b')$. Therefore, $\delta(b) = \phi(b')$ and there exists $b'' \in B_3$ such that $\psi(b'') = \phi(b')$ and $b'' \geq b'$. This shows $\delta(B_1) \leq \psi(B_3)$ and hence $B_1 \leq^\flat B_3$.

To show that $\preccurlyeq^\flat$ is anti-symmetric, let $B_1 \preccurlyeq^\flat B_2$ and $B_2 \preccurlyeq^\flat B_1$. As was shown above, there exist a, ϕ and ψ such that $\alpha(B_1) \leq^\flat \phi(B_2) \leq^\flat \psi(B_1)$. In particular, if we define $g : \alpha(B_1) \to \psi(B_1)$ by $g(b, n) = (b', n)$ where $\psi(b') = n$, it is easy to see that g is one-to-one and monotone and, since B_1 is finite, it is the identity map. If $b'' \in B_2$ and $\phi(b'') = n$, then $b \leq b'' \leq b' = b$, so $b = b''$ where $\alpha(b) = \psi(b') = n$. Therefore, every element of B_1 is in B_2 and vice versa, i.e. $B_1 = B_2$. This shows that $\preccurlyeq^\flat$ is a partial order.

Since $B_1 \rightsquigarrow B_2$ implies $B_1 \preccurlyeq^\flat B_2$, we conclude $\trianglelefteq \subseteq \preccurlyeq^\flat$. Conversely, if $B_1 \preccurlyeq^\flat B_2$, i.e. $\phi(B_1) \leq^\flat \psi(B_2)$, then, according to [22], $\psi(B_2)$ can be obtained from $\phi(B_1)$ by a sequence of $\mapsto$ updates which, if we drop indices, are translated into $\rightsquigarrow$ updates on bags. Therefore, $B_1 \trianglelefteq B_2$, which proves $\trianglelefteq = \preccurlyeq^\flat$. □

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

Proposition 5.2 *There exists an $O(n^{5/2})$ time complexity algorithm that, given two bags B_1 and B_2 in $\mathcal{P}^b_{\text{fin}}(D)$, returns* true *if $B_1 \trianglelefteq B_2$ and* false *otherwise.*

Proof. Given B_1 and B_2, consider two labelings ϕ and ψ on B_1 and B_2. Assume without loss of generality that the codomains of ϕ and ψ are disjoint. Define a bipartite graph $G = \langle V, E \rangle$ by $V := \phi(B_1) \cup \psi(B_2)$ and $E := \{((b, n), (b', n)) \mid (b, n) \in \phi(B_1), (b', n') \in \psi(B_2), b \leq b'\}$. It can be easily concluded from proposition 5.1 that $B_1 \trianglelefteq B_2$ iff there is a matching in G that contains all $\phi(B_1)$. In other words, $B_1 \leq B_2$ iff the cardinality of the maximal matching in G is that of B_1. The proposition now follows from the facts that all maximal matching in G have the same cardinality (as bases of a matroid) and that the Hopcroft-Karp algorithm finds a maximal matching in $O(n^{5/2})$ where $n = |V|$ (see [28]). □

There is a big difference between orders on sets and bags. While $X \leq^\flat Y$ does not say anything about cardinality of X and Y, $B_1 \trianglelefteq B_2$ implies that the cardinality of B_1 is less than or equal to the cardinality of B_2. This reflects our point of view that having a bag rather than a set stored in a database means that each element of a bag represents an object and having two or more occurrences of the same elements means that at the moment some objects are indistinguishable. Therefore, the cardinality can not be reduced in the process of obtaining more information. In particular, in the set case the lower powerdomain ordering can be obtained as the transitive-reflexive closure of the following binary relation:$A \mapsto (A - A') \cup \{a\}$ where

$a \geq a'$ for all $a' \in A'$ and $A \mapsto A \cup \{a\}$. However, applying the same idea to bags amounts to the loss of information about the number of occurrences of each element in a bag. In particular, similar transformations when applied to bags give us the ordering $\sqsubseteq$ which was used in the proof of proposition 3.4.

In the remainder of this section we propose another technique for extending a partial order to bags. However, our motivation this time is different. We are no longer interested in having an order whose intuitive meaning is being more informative, but rather an order which could be easily implemented and used for various purposes, especially designing effective physical data organization for sets and bags. For such purposes, it is important to have a total (linear) order. In particular, sorting algorithms and duplicate detection and elimination algorithms rely on linear orders. However, it can be easily seen that $\trianglelefteq$ is not a linear order even if the underlying order $\leq$ is: suppose $a \leq b$; then $\{\!|a, a|\!\} \not\trianglelefteq \{\!|b|\!\}$ and $\{\!|b|\!\} \not\trianglelefteq \{\!|a, a|\!\}$.

To design a linear order on bags, it is easier to work with the set representation via theorem 4.4. The following lemma is the key step:

Lemma 5.3 *Let $\langle D, \leq \rangle$ be a partially ordered set. Define an order $\lesssim^\flat$ on the finite subsets of D as follows:*

$$X \lesssim^\flat Y \text{ iff either } X \leq^\flat Y \text{ and } Y \not\leq^\flat X, \text{ or } X \leq^\flat Y \text{ and } Y \leq^\flat X \text{ and } X - Y \leq^\flat Y - X.$$

Then $\lesssim^\flat$ is a partial order. Moreover, if $\leq$ is a linear order, then so is $\lesssim^\flat$.

Proof. We proceed by cases on X and Y. The cases where X or Y is empty are trivial. So let both X and Y be nonempty. Let us write $X \approx^\flat Y$ to mean $X \leq^\flat Y$ and $Y \leq^\flat X$. Now observe that if $X \approx^\flat Y$ implies $X \cap Y \neq \emptyset$. Indeed, if $X \cap Y = \emptyset$, $x \leq y$ for $x \in X$ and $y \in Y$ means $x \lneqq y$ and then $X \approx^\flat Y$ implies the existence of an infinitely increasing chain $x_1 \lneqq y_1 \lneqq x_2 \lneqq y_2 \lneqq \ldots$ which contradicts the assumption that X and Y are finite. This also implies asymmetry of $\lesssim^\flat$: if $X \lesssim^\flat Y$ and $Y \lesssim^\flat X$, then $X - Y \approx^\flat Y - X$, i.e. $X - Y = Y - X = \emptyset$ and therefore $X = Y$. To prove transitivity, assume $X \lesssim^\flat Y$ and $Y \lesssim^\flat Z$. Then $X \leq^\flat Z$. If $Z \not\leq^\flat Y$ or $Y \not\leq^\flat X$, then easily $Z \not\leq^\flat X$ and $X \lesssim^\flat Z$. Assume $Z \leq^\flat Y$ and $Y \leq^\flat X$. Then $X \approx^\flat Z$, $X - Y \leq^\flat Y - X$ and $Y - Z \leq^\flat Z - Y$. We must prove $X - Z \leq^\flat Z - X$. Let $x \in X - Z$. Two cases arise.

Case $x \notin Y$. Then $x \in X - Y$ and there exists $y_1 \in Y - X$ such that $x \leq y_1$. If $y_1 \in Z$, we are done; otherwise $y_1 \in Y - Z$ and there exists $z_1 \in Z - Y$ such that $z_1 \geq y_1$. If $z_1 \notin X$, we are done. If $z_1 \in X$, then $z_1 \in X - Y$ and we continue the process. If at some step an element of $Z - X$ is found that is greater than x, we are done. Otherwise, we arrive at an infinite chain $x \leq y_1 \leq z_1 \leq y_2 \leq z_2 \leq \ldots$, and since every two neighbours in this chain come from disjoint sets, it is an infinitely increasing chain of element of $X \cup Y \cup Z$, which contradicts the assumption that all sets are finite.

Case $x \in Y$. Then $x \in Y - Z$ and we find $z_1 \in Z - Y$ such that $z_1 \geq x$. If $z_1 \notin X$, then $z_1 \in Z - X$ and we are done. If $z_1 \in X$, then $z_1 \in X - Y$. Now the method of case 1 applies which finishes the proof of case 2. Thus, $\lesssim^\flat$ is a partial order.

Now assume that $\leq$ is a linear order. Then $X \not\leq^\flat Y$ implies $Y \leq^\flat X$ (because there is $x \in X$ such that $x \not\leq y$ for all y, i.e. $x \geq y$). The linearity of $\lesssim^\flat$ follows immediately. □

This linear order on sets is equivalent to one of the three orderings which Kupert, Saake, and Wegner used in their study of duplicate elimination algorithms for nonflat relational databases [20]. However, they did not prove the linearity of their orderings. Also their formulation, which makes explicit use of cardinality and minimum value, is somewhat less elegant than the formulation given above. This linear order is very pleasant — it can be expressed in our very limited nested relational language!

Theorem 5.4 *Suppose a linear order $\leq_b$ is given for every base type b. Then a linear order $\leq_t$ can be computed by $\mathcal{NRL}(\mathbf{N}, \sum, \cdot, \dot{-}, +, bool, cond, =)$ for each type t.*

Proof. First the linear order $\leq_t$ is defined by induction on t as follows. For base types, $\leq_b$ is as given. For pairs, the lexicographic order is used. It is easy to see that it is linear if the components are linearly ordered. Finally for sets, the order $\lesssim^b$ in lemma 5.3 is used. One particular implementation in $\mathcal{NRL}(=, \mathbf{N}, \cdot, \dot{-}, +, \sum, cond)$ is given in the appendix. □

We end this section with an interesting corollary on one of the most important uses of linear ordering. Let $sort : \{s\} \to \{s \times \mathbf{N}\}$ be a sorting function. That is, $sort(R) = \{(o_1, 1), \ldots, (o_k, k)\}$ where $R = \{o_1, \ldots, o_k\}$ and $o_1 < \ldots < o_k$. Then

Corollary 5.5 *sort is definable in $\mathcal{NRL}(\mathbf{N}, \sum, \cdot, \dot{-}, +, bool, cond, =)$.*

Proof. $sort(R) := \{(r,\ \Sigma\{\mathit{if}\ c \leq r\ \mathit{then}\ 1\ \mathit{else}\ 0 \mid c \in R\}) \mid r \in R\}$. □

6 Conservative extension property for set languages with aggregate functions

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. 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.

As seen earlier, bags give rise to natural numbers. In this section, two conservative extension results are presented. First, the language $\mathcal{NRL}(\mathbb{Q}, \sum, \cdot, +, \div, cond, eq)$ obtained by adding rational arithmetics to $\mathcal{NRL}$ is shown to have the conservative extension property with 0 as the fixed constant. Second, $\mathcal{NRL}(\mathbf{N}, \sum, \cdot, +, \dot{-}, cond, eq)$, and hence $\mathcal{NBL}(monus, unique)$, has it too. These results allow us to reduce questions about bags to questions about arithmetics in subsequent sections.

Queries such as "select count from column," "select maximum from column," and "select minimum from column" are expressible in $\mathcal{NRL}(\mathbf{N}, \sum, \cdot, +, \dot{-}, eq)$. However, in order to deal with queries such as "select average from column" and "select variance from column" a division primitive is needed. A natural extension is to augment the nested relational language with rational division: $\mathcal{NRL}(\mathbb{Q}, \sum, \cdot, +, \dot{-}, \div, eq)$. This language has the conservative extension property. For convenience, we prove it on the equivalent language $\mathcal{NRC}(\mathbb{Q}, \sum, \cdot, +, \dot{-}, \div, cond, eq_b)$ where the booleans are interpreted by 1 (true) and 0 (false), equality tests on base types only are provided, and the conditional construct is available.

Theorem 6.1 *Let $e : s$ be an expression of $\mathcal{NRC}(\mathbb{Q}, \sum, \cdot, +, \dot{-}, \div, cond, eq_b)$. 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 (sketch). The proof is adapted from Wong [38]. The idea is to design a rewrite system which eliminates the construction of intermediate data structures and then show that any normal form induced by

the rewrite system can serve as the required e'. The key rules are given below; the full proof is relegated to the addendum.

- $\Sigma\{e \mid x \in \{\}\} \leadsto 0$
- $\Sigma\{e \mid x \in \{e'\}\} \leadsto e[e'/x]$
- $\Sigma\{e \mid x \in e_1 \cup e_2\} \leadsto \Sigma\{e \mid x \in e_1\} + \Sigma\{\mathit{if}\ y \notin e_1\ \mathit{then}\ e[y/x]\ \mathit{else}\ 0 \mid y \in e_2\}$
- $\Sigma\{e \mid x \in \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3\} \leadsto \mathit{if}\ e_1\ \mathit{then}\ \Sigma\{e \mid x \in e_2\}\ \mathit{else}\ \Sigma\{e \mid x \in e_3\}$
- $\Sigma\{e \mid x \in \bigcup\{e_1 \mid y \in e_2\}\} \leadsto \Sigma\{\Sigma\{(e \div \Sigma\{\Sigma\{x = v \mid v \in e_1[u/y]\} \mid u \in e_2\}) \mid x \in e_1\} \mid y \in e_2\}$.

The last rule above deserves special attention. Consider the *incorrect* equation: $\Sigma\{e \mid x \in \bigcup\{e_1 \mid y \in e_2\}\} = \Sigma\{\Sigma\{e \mid x \in e_1\} \mid y \in e_2\}$. 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 evaluates 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 rewrite rule is used to handle duplicates properly. □

In the next section, the inexpressibility of division, enumeration of natural numbers from 0 to n for a given n, etc. in $\mathcal{NBL}(\mathit{monus}, \mathit{unique})$ is established as a consequence of the above theorem. Observe that $\div$ is critical for achieving conservative extension in the nested relational language endowed with rationals. Therefore, it is possible that $\mathcal{NBL}(\mathit{monus}, \mathit{unique})$ may not possess the conservative extension property. Fortunately, this is not the case because $\mathcal{NBL}(\mathit{monus}, \mathit{unique})$ has sufficient horsepower to compute a linear order at all types. In the next theorem, the linear order defined in theorem 5.4 is exploited to show that functions in $\mathcal{NRC}(=, \mathbf{N}, \cdot, \dot{-}, +, \sum, \mathit{cond})$, and hence $\mathcal{NBL}(\mathit{monus}, \mathit{unique})$, do not depend on intermediate data structures.

Theorem 6.2 *Let $e : s$ be an expression of $\mathcal{NRC}(\mathbf{N}, \sum, \cdot, +, \dot{-}, \mathit{cond}, =)$. 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 rewrite rule involving $\div$ by the rule: $\Sigma\{e \mid x \in \bigcup\{e_1 \mid y \in e_2\} \leadsto \Sigma\{\Sigma\{(\mathit{if}\ (\Sigma\{(\mathit{if}\ x \in e_1[w/x] \wedge w \neq y \wedge w \leq y\ \mathit{then}\ 1\ \mathit{else}\ 0 \mid w \in e_2\}) = 0\ \mathit{then}\ e\ \mathit{else}\ 0) \mid x \in e_1\} \mid y \in e_2\}$. The idea behind this rule is that if several items y_i in e_2 produce e_1 with some common x_j, then the linear ordering is used to pick the x_j generated by the smallest y_i. The details can be found in the appendix. □

Let us prove an easy but interesting corollary of theorem 6.2. Consider the following additional primitives:

$$\frac{}{TC^s : \{s \times s\} \to \{s \times s\}} \qquad \frac{g : \{s\} \quad f : \{s\} \to \{s\}}{\mathit{bfix}(f, g)^s : \{s\}} \qquad \frac{}{\mathit{powerset}^s : \{s\} \to \{\{s\}\}}$$

where $TC(R)$ is the transitive closure of R; $\mathit{bfix}(f, g)$ is the bounded fixpoint of f with respect to g, that is, the fixpoint of the equation $f(R) = g \cap (R \cup f(R))$; and $\mathit{powerset}(R)$ is the powerset of R. It should be remarked here that TC and bfix do not take the language out of polynomial time.

Corollary 6.3 *$\mathcal{NRC}(\mathbf{N}, \sum, \cdot, +, \dot{-}, \mathit{cond}, \mathit{bool}, =, TC)$, $\mathcal{NRC}(\mathbf{N}, \sum, \cdot, +, \dot{-}, \mathit{cond}, \mathit{bool}, =, \mathit{bfix})$, and $\mathcal{NRC}(\mathbf{N}, \sum, \cdot, +, \dot{-}, \mathit{cond}, \mathit{bool}, =, \mathit{powerset})$ have the conservative extension property. The first two have it with fixed constant 1 and the last one has it with fixed constant 2.*

Proof. We provide the proof for the first one, the other two are straight forward adaptation of the same technique. First observe that $\mathcal{NRC}(\mathbb{N}, \sum, \cdot, +, \dot{-}, cond, bool, =, TC^{\mathbb{N}})$, where we restrict computation of transitive closure to binary relations of natural numbers, has the conservative extension property with constant 1. This follows from theorem 6.2. Therefore, it suffices for us to show that TC^s is expressible in it for any s. This can be achieved by exploiting the *sort* function given in corollary 5.5 by defining $TC(R) := decode(TC^{\mathbb{N}}(encode(R, sort(dom(R)))), sort(dom(R)))$, where

- $dom(R) := \{x \mid (x,y) \in R\} \cup \{y \mid (x,y) \in R\}$
- $encode(R,C) := \{(a,b) \mid (r,s) \in R, (r,a) \in C, (s,b) \in C\}$
- $decode(R,C) := \{(r,s) \mid (a,b) \in R, (r,a) \in C, (s,b) \in C\}$ □

Conservative extension property was first studied by Paredaens and Van Gucht [29] and later by Van den Bussche [10]. They proved that $\mathcal{NRC}(eq)$ has it when input and output are restricted to flat relations. It was then extended by Wong [38] to any input and output. More recently, Suciu [32] managed to prove the remarkable theorem that $\mathcal{NRC}(eq, bfix)$, note the absence of natural numbers, has the conservative extension property when input and output are restricted to flat relations. 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 [12] obtained a noncollapsing hierarchy theorem. Let $gen : \mathbb{N} \to \{\mathbb{N}\}$ 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 $\mathcal{NRC}(\mathbb{N}, \sum, \cdot, +, \dot{-}, cond, bool, =, gen, powerset)$ 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.

By the conservative extension property, the class of functions on flat relations computed by $\mathcal{NRL}(\mathbb{Q}, \sum, \cdot, +, \dot{-}, \div, bool, cond, =)$ is precisely that computed by flat relational algebra endowed with the same primitives. This has a practical significance because it implies that $\mathcal{NRL}(\mathbb{Q}, \sum, \cdot, +, \dot{-}, \div, bool, cond, =)$ can be used as a convenient interface to databases that speak SQL. A theoretically more interesting consequence is that every function of type $\{\!|unit|\!\} \to \{\!|unit|\!\}$ in $\mathcal{NBL}(monus, unique)$ corresponds to very simple arithmetics. This will be exploited in the next section where arithmetic properties of the bag query languages are studied.

7 Relationship between bags and numbers

As seen earlier, natural numbers are present in our yardstick query language as objects of type $\{\!|unit|\!\}$. That made it possible to translate $\mathcal{NBL}(monus, unique)$ into *set* language augmented by either rational or natural numbers and some arithmetic. In this section we use the conservative extension results from the previous section to describe a (very limited) arithmetics of bag languages. We show that no property of natural numbers that is simultaneously infinite and co-infinite can be tested in either language. This is particularly surprising for the language augmented by rational numbers and *division*, since it implies 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 arithmetic functions in $\mathcal{NBL}(monus, unique)$.

Theorem 7.1 *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 membership test for $\mathcal{U}$ can not be expressed in $\mathcal{NRL}(\mathbb{Q}, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, =)$.*

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 can not occur in the course of evaluation of e.

First we establish a useful property of expressions of type $\mathbb{Q} \to \mathbb{Q}$ that are defined everywhere. We say that an expression e of type $\mathbb{Q} \to \mathbb{Q}$ is a plus-expression if there is a number n depending on e such that for any $x \geq n$: $e(x) > 0$ and it is a zero-expression if there is a number n depending on e such that for any $x \geq n$: $e(x) = 0$. Now our goal is to prove that any expression e of type $\mathbb{Q} \to \mathbb{Q}$ that is defined everywhere is either plus- or zero-expression. In fact, we prove a stronger result that shows in addition that for any plus-expression there are two polynomial functions with rational coefficients $p(x)$ and $q(x)$ such that for any $x \geq n$: $e(x) = p(x)/q(x)$ and $p(x), q(x) \geq 0$.

Let e be of type $\mathbb{Q} \to \mathbb{Q}$. Since $\mathcal{NRL}(\mathbb{Q}, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, =)$ has the conservative extension property, 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 $\mathbb{Q} \to \mathbb{Q}$.

Now we prove our main claim by induction on the structure of e. The base case is immediate since every constant is either plus- or zero-expression. Let $e = e_1 + e_2$. If both e_1 and e_2 are zero-expressions, then so is e. If 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, then e is a plus-expression given by $(p_1(x) \cdot q_2(x) + p_2(x) \cdot q_1(x))/(q_1(x) \cdot q_2(x))$ for $x \geq \max\{n_1, n_2\}$. If one of the subexpressions is a plus-expression and the other one is a zero-expression, then e is a plus-expression whose polynomial representation coincides with the representation of the plus-subexpression for $x \geq \max\{n_1, n_2\}$. The argument for $e = e_1 \cdot e_2$ and $e = e_1 \div e_2$ is similar. 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. 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 for every $y > x_f$: $f(y) > 0$, then for every $x \geq \max\{n_1, n_2, x_f\} + 1$, $p_1(x)/q_1(x) - p_2(x)/q_2(x) > 0$ 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 for any $y > x_f$: $f(y) < 0$, then for every $x \geq \max\{n_1, n_2, x_f\} + 1$, $p_1(x)/q_1(x) - p_2(x)/q_2(x) < 0$ and therefore $e(x) = 0$, i.e. e is a zero-expression. That finishes the proof of our claim.

Now observe that if e is a test for the $\mathcal{U}$-membership, it is neither plus- nor zero-expression. Thus, it can not be expressed in $\mathcal{NRL}(\mathbb{Q}, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, =)$. □

It is well known that the traditional relational languages cannot express parity test [8]. By the result of [38], it cannot be expressed in $\mathcal{NRL}(eq)$. It follows from the theorem we just proved that it remains

inexpressible even in the greatly enhanced $\mathcal{NRL}(\mathbb{Q}, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, =)$ and hence not expressible in $\mathcal{NBL}(monus, unique)$. This is another consequence of conservative extension.

Corollary 7.2 *Parity test is not expressible in* $\mathcal{NRL}(\mathbb{Q}, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, =)$. □

The above corollary 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, $\mathcal{NBL}(monus, unique)$ can not 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.

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

Proposition 7.3 *Let* $\leq$ *be the usual order on* $\mathbb{Q}$. *Then it can not be expressed in* $\mathcal{NRL}(\mathbb{Q}, \Sigma, \cdot, +, \div, bool, cond, =)$.

Proof. It is enough to show that the following function from $\mathbb{Q}$ to $\mathbb{Q}$ can not be expressed: $g(x) = 0$ if $x \leq 1$ and $g(x) = 1$ if $x > 1$. It follows from conservative extension that it suffices to show that g can not be defined using $+, \cdot, \div, =$, if-then-else and constants. We proceed by proving the following claim: 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, i.e. $g(x) \neq p(x)/q(x)$ for only finitely many $x \in \mathbb{Q}$. We show it 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. It shows that g coincides with either p_1/q_1 or p_2/q_2 almost everywhere.

Now, if f is expressible in the language, it must also coincide with a ratio of two polynomials p and q almost everywhere. Since p has infinitely many roots, it must be identical to zero, which contradicts our assumption that $p(x)/q(x) = 1$ for almost all $x > 1$. This contradiction shows that $\leq$ is not expressible. □

We now turn to the nested relational language $\mathcal{NRL}(\mathbb{N}, \Sigma, \cdot, +, \dot{-}, bool, cond, =)$ which is equivalent to $\mathcal{NBL}(monus, unique)$. A unary function $f : \mathbb{N} \to \mathbb{N}$ is said to be *almost polynomial* if there exists a polynomial function $g : \mathbb{N} \to \mathbb{N}$ (that is, a function built from its argument and constants by using addition, subtraction and multiplication) and a number n such that for any $x \geq n$ it holds: $f(x) = g(x)$ (that is, f is g in all but finitely many points). The class of almost polynomial functions is denoted by $\mathcal{P}^{\approx}$.

Proposition 7.4 $\mathcal{P}^{\approx}$ *is the class of unary arithmetic functions expressible in* $\mathcal{NBL}(monus, unique)$.

Proof. The class of unary arithmetic functions (that is, functions of type $\{|unit|\} \to \{|unit|\}$) coincides with the class of functions of type $\mathbb{N} \to \mathbb{N}$ in $\mathcal{NRL}(\mathbb{Q}, \Sigma, \cdot, +, \div, \dot{-}, bool, cond, =)$ which, according to the

conservative extension result, are built from the variables and constants by using addition, multiplication and modified subtraction ($\dot{-}$). We prove that each of these functions is in $\mathcal{P}^{\approx}$ by induction on the structure of the function. The cases for constants and variables are immediate. The addition and multiplication cases are similar to the proof of theorem 7.1. Let $f = f_1 \dot{-} f_2$ where f_i coincides with a polynomial g_i for $x \geq n_i$, i=1,2. Let x_f be the maximal root of $g_1 - g_2$ and $n = \max\{n_1, n_2, x_f\} + 1$. Then there are two cases. If $g_1(x) - g_2(x) \leq 0$ for each $x \geq n$, then $f(x) = 0$ for $x \geq n$ and 0 is the polynomial representing f for $x \geq n$. Or, if $g_1(x) - g_2(x) > 0$ for $x \geq n$, then $g_1 - g_2$ is the polynomial representing f.

Conversely, any almost polynomial function can easily be expressed in $\mathcal{NRL}(\mathbf{N}, \Sigma, \cdot, +, \dot{-}, bool, cond, =)$. Proposition is proved. □

Corollary 7.5 *None of the following functions is expressible in* $\mathcal{NBL}(monus, unique)$*:*

- *parity test;*
- *division by a constant;*
- *bounded summation;*
- *bounded product;*
- $f : \mathbf{N} \to \{\!|\mathbf{N}|\!\}$ *such that* $f(n) = \{\!|0, 1, \ldots, n|\!\}$.

Proof. That parity test is not expressible follows either from theorem 7.1 or the previous proposition. Suppose $div_2(n) = n/2$ (integer division) 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 $\mathcal{NRL}(\mathbf{N}, \Sigma, \cdot, +, \dot{-}, bool, cond, =)$ are polynomial, the size of the output of any function $f : \mathbf{N} \to t$ is bounded by a constant (since the size of the input is 1) which proves inexpressibility of the generator of smaller numbers. □

Therefore, the arithmetic of our ambient query language is very limited. In the next section where non-polynomial primitives are studied, we show that two extended languages give rise to all elementary and primitive recursive functions respectively.

8 Power operators, bounded loop and structural recursion

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

In [5], it was shown that endowing $\mathcal{NRL}(eq)$ with a structural recursion primitive, which we denote by *s_sri*, or with the *powerset* operator yields languages that are equi-expressive. However, this is contingent upon the highly 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,

Proposition 8.1 $\mathcal{NRL}(eq) \simeq \mathcal{NRL}(eq, s\_sri) \simeq \mathcal{NRL}(eq, powerset)$, *if all types have finite domains.* □

Hence the extra power of *s_sri* and *powerset* has effect only when there are types whose domains are infinite. Types such as natural numbers proved to be important in the earlier part of this report. Therefore, the relationship of structural recursion and power operators should be re-examined.

The syntax for the structural recursion construct on sets is

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

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

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

This section has three main objectives. First, we endow $\mathcal{NBL}(monus, unique)$ with the bag analogs of the powerset and structural recursion operators and we show that the former is strictly less expressive than the latter. Second, we suggest an efficient bounded loop primitive which captures the power of structural recursion but does not require any preconditions. Finally, we characterize functions on natural numbers that can be expressed in languages endowed with powerbag and structural recursion as elementary and primitive recursive functions respectively.

8.1 Powerset, powerbag and structural recursion

Grumbach and Milo [12], following Abiteboul and Beeri [1], introduced the *powerbag* operator into their nested bag language. The semantics of *powerbag* is the function that produces a bag of all subbags of the input bag. For example, $powerbag\{\!|1, 1, 2|\!\} = \{\!|\{\!|\}\!|, \{\!|1|\!\}, \{\!|1|\!\}, \{\!|2|\!\}, \{\!|1, 1|\!\}, \{\!|1, 2|\!\}, \{\!|1, 2|\!\}, \{\!|1, 1, 2|\!\}|\!\}$. They also defined the *powerset* operator on bags as $unique \circ powerbag$. For example, $powerset\{\!|1, 1, 2|\!\}$ is $\{\!|\{\!|\}\!|, \{\!|1|\!\}, \{\!|2|\!\}, \{\!|1, 1|\!\}, \{\!|1, 2|\!\}, \{\!|1, 1, 2|\!\}|\!\}$. We do not consider *powerset* on bags further because

Proposition 8.2 $\mathcal{NBL}(monus, unique, powerbag) = \mathcal{NBL}(monus, unique, powerset)$.

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 $\mathcal{NBL}(monus, unique, powerset)$ 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. Apply *powerset* to this bag followed by elimination of labels produces $powerbag(B)$. □

Structural recursion on bags is defined using the construct

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

It is required that i satisfy the commutativity precondition: $i(a, i(b, X)) = i(b, i(a, X))$, which can not be automatically verified [4]. It 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 8.3 *Let $f : s \to t$ be an expression of $\mathcal{NBA}(monus, unique, powerset)$. 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 e. For any polynomial operator p in $\mathcal{NBL}(monus,\ unique)$, it is safe to define c_p to be 1. For operators that are not polynomial, define $c_{powerset} := 1$, $c_{\langle f,g \rangle} := \max(c_f, c_g)$, $c_{f \circ g} := c_f + c_g$, and $c_{b\_map(f)} := 1 + c_f$. □

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

Theorem 8.4 $\mathcal{NBL}(powerbag, monus, unique) \subsetneq \mathcal{NBL}(b\_sri, monus, unique)$.

Proof. Inclusion is easy [5]. To prove strictness, define an auxiliary function $g : \{|unit|\} \to \{|unit|\}$ in $\mathcal{NBL}(b\_sri, monus, unique)$ 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 8.3, f can not be expressed in $\mathcal{NBL}(powerbag, monus, unique)$. □

8.2 Bounded loop and structural recursion

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

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

Its semantics is as follows: $loop(f)(\{\!|o_1,\ldots,o_n|\!\},o) = f(\ldots f(o)\ldots)$ where f is applied n times to o. Remark that it is enough to consider only $loop^{unit}(f) : \{\!|unit|\!\} \times s \to s$.

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 8.5 $\mathcal{NBL}(monus, unique, loop) \simeq \mathcal{NBL}(monus, unique, b\_sri)$.

Proof. For the $\mathcal{NBL}(monus, unique, loop) \subseteq \mathcal{NBL}(monus, unique, b\_sri)$ part, it suffices to observe that $loop(f)(n,e) = b\_sri(f \circ \pi_2,\ e)(n)$. The $\mathcal{NBL}(monus, unique, b\_sri) \subseteq \mathcal{NBL}(monus, unique, loop)$ 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 $\mathcal{NBL}(monus, unique, loop)$. □

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 8.5 has exponential complexity but the source of inefficiency is in computing all permutations in order to return all possible outcomes. If we allow to 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 corollary 5.5). 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 $o_1 \le \ldots \le o_k$ in the linear order of theorem 5.4.

8.3 Arithmetic properties of non-polynomial languages

In this section functions on natural numbers expressible in $\mathcal{NBL}(monus,\ unique,\ loop)$ are characterized. Before proving the two theorems, let us argue that they are very intuitive and are not unexpected. There are two classical results in recursion theory [25]. 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 [25]. In view of these results and the fact the *loop* construct is just a *for–do* iteration, the following result is very natural.

Theorem 8.6 *The class of functions $f : \mathbf{N} \times \ldots \times \mathbf{N} \to \mathbf{N}$ definable in $\mathcal{NBL}(monus, unique, loop)$ coincides with the class of primitive recursive functions.*

Grumbach and Milo showed [12] that their bag language, which is equivalent to $\mathcal{NBL}(monus, unique, powerbag)$, 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 [30]. That is, the following functions are in $\mathcal{E}$ if g is in $\mathcal{E}$:

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

Using techniques different from [12], we prove the following:

Theorem 8.7 *The class of functions $f : \mathbf{N} \times \ldots \times \mathbf{N} \to \mathbf{N}$ definable in $\mathcal{NBL}(monus, unique, powerbag)$ coincides with the class of Kalmar-elementary functions.*

Let us first give the proof of theorem 8.6.

Proof of theorem 8.6. Throughout the proof we use $\mathbf{N}$ as abbreviation for $\{\!|unit|\!\}$ and n as an abbreviation for $\{\!|(), \ldots, ()|\!\}$ (n times). We start with theorem 8.6. 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 8.4. Therefore, encoding and decoding functions for tuples can be expressed. In view of that and the Robinson-Gladstone result [25], to prove that all primitive recursive functions can be computed by $\mathcal{NBL}(monus, unique, loop)$, 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 $\mathcal{NBA}(monus, unique, loop)$ there is a monotone primitive recursive function φ_e of one argument such that $\mathsf{size}\, e(o) \leq \varphi_e(\mathsf{size}\, o)$. 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 $\mathsf{size}\, d' = \sum_i \mathsf{size}\, o'_i \leq \sum_i \varphi_f(\mathsf{size}\, o_i) \leq \sum_i \varphi_f(\mathsf{size}\, d) \leq \mathsf{size}\, d \cdot \varphi_f(\mathsf{size}\, d)$. 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 operations of $\mathcal{NBA}(monus, unique, loop)$ into computations on a Turing machine shows that the space complexity for every expression in the language remains bounded by a primitive recursive functions. Therefore, if $f : \mathbf{N} \times \ldots \times \mathbf{N} \to \mathbf{N}$ is a function computable $\mathcal{NBL}(monus, unique, loop)$, 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. □

Next we give the

Proof of theorem 8.7. First we show that bounded sum and bounded product are expressible in $\mathcal{NBL}(monus, unique, powerbag)$. 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 expressibility of f_2 resembles the proof of expressibility of the α primitive of [22] 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 $\mathcal{NBL}(monus, unique, powerbag)$ is bounded above by $hyper(c, n)$ where c is a constant, see proposition 8.3. 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 $f : \mathbf{N} \times \ldots \times \mathbf{N} \to \mathbf{N}$ is computable in $\mathcal{NBL}(monus, unique, powerbag)$, it can be computed by a Turing machine whose space complexity is bound by a function from $\mathcal{E}$. Whence $f \in \mathcal{E}$, see [23]. This finishes the proof of theorem 8.7. □

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 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 8.8 *$\mathcal{NBL}(monus, unique, iter, powerset^{unit})$ 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 [30, 1.4]. It is straightforward to verify that all additional initial functions can be expressed in the presence of $powerset^{unit}$. □

9 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. An ordering for dealing with partial information in bags is given and a technique for lifting linear order on base types to linear order on all types is presented. The linear order is used to prove that our bag languages are conservative with respect to height of input and output. The relationship between bags and natural numbers is studied via the conservative extension property. In particular, we showed that properties that are simultaneously infinite and co-infinite are inexpressible. 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 obtained by Breazu-Tannen, Buneman, and Wong on sets [5]. These two reports taken together are a foundation for programming with collection types using the paradigm of monad and structural recursion first investigated by Breazu-Tannen, Buneman, and Naqvi [3].

Future work. There are many further problems which we would like to investigate. Many properties of the partial order on bags proposed to deal with incomplete information remain to be explored. For instance, its role in the theory of powerdomains remains to be seen. Powerdomains are typically used to give semantics for collections in programming languages, see [7, 14, 21]. We also would like to see how the idea of using approximations to model partial information [6, 13] can be extended to bags.

It is known that the presence of a linear order adds tremendous power to first-order query languages [16, 34]. 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. Also, other kinds of linear orders on nested collections such as those in [20] should be studied.

There are several conjectures we have not yet proved. Does adding primitive $gen : \mathbf{N} \to \{\mathbf{N}\}$ that produces numbers from 0 to n for a given input n gives us precisely lower elementary functions [30]? Are functions such as transitive closure or test for balanced binary trees expressible in the set language equivalent to $\mathcal{NBL}(monus, unique)$? What is the expressive power of this set language augmented by transitive closure? We know that test for balanced binary trees can be expressed in this language, but can it express bounded fixpoint?

We were able to demonstrate the conservative extension property for the nested set language with aggregate functions and additional primitives such as transitive closure, bounded fixpoint and powerset by reducing these primitives to the corresponding ones on natural numbers. What is the general property of these primitives that allowed this reduction?

Breazu-Tannen, Buneman and Wong [5], Libkin and Wong [22], and this paper studied the use of monads and structural recursion for querying sets, or-sets and bags respectively. We hope to extend this methodology to other collection types such as lists, arrays, etc.

Acknowledgements. Peter Buneman gave us the initial inspiration and provided many helpful suggestions. Val Breazu-Tannen and Dan Suciu helped us understand programming with collection types. We also thank Jean Gallier and Scott Weinstein for answering many of our questions and Paul Taylor for the diagram macros.

References

[1] S. Abiteboul and 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] J. Albert. Algebraic properties of bag data types. In *Proceedings of 17th International Conference on Very Large Databases*, pages 211–219, 1991.

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

[4] V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with sets/bags/lists. In *LNCS 510: Proceedings of 18th International Colloquium on Automata, Languages, and Programming, Madrid, Spain, July 1991*, pages 60–75. Springer Verlag, 1991.

[5] V. Breazu-Tannen, P. Buneman, and L. Wong. Naturally embedded query languages. In J. Biskup and R. Hull, editors, *LNCS 646: Proceedings of International Conference on Database Theory, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 1992. Full paper available as UPenn Technical Report MS-CIS-92-47.

[6] P. Buneman, S. Davidson and A. Watters, A semantics for complex objects and approximate answers, *Journal of Computer and System Sciences*, 43:170–218, 1991.

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

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

[9] L. S. Colby. A recursive algebra for nested relations. *Information Systems*, 15(5):567–582, 1990.

[10] 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.

[11] J. Van den Bussche and J. Paredaens. The expressive power of structured values in pure OODB. Technical Report 90-23, University of Antwerp, Department of Mathematics and Computer Science, Universiteitsplein 1, B-2610 Antwerp, Belgium, November 1990. Revised version appeared in proceedings of PODS'91.

[12] S. Grumbach and T. Milo. Towards tractable algebras for bags. In *Proceedings of 12th ACM Symposium on Principles of Database Systems*, Washington, D. C., May 1993. To appear.

[13] C. Gunter, The mixed powerdomain, *Theoretical Computer Science*, 103:311–334, 1992.

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

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

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

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

[18] A. Klausner and N. Goodman. Multirelations: semantics and languages. In A. Pirotte and Y. Vassiliou, editors, *Proceedings of 11th International Conference on Very Large Databases, Stockholm, August 1985*, pages 251–258, Los Altos, CA, August 1985. Morgan Kaufmann.

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

[20] K. Kupert, G. Saake, and L. Wegner. Duplicate detection and deletion in the extended NF^2 data model. In W. Litwin and H. J. Schek, editors, *LNCS 367: Foundation of Data Organization and Algorithms*, pages 83–101. Springer-Verlag, June 1989.

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

[22] L. Libkin and 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. To appear. Full paper available as UPenn Technical Report MS-CIS-92-88.

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

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

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

[26] A. Ohori, P. Buneman, and V. Breazu-Tannen. Database programming in Machiavelli: a polymorphic language with static type inference. In James Clifford, Bruce Lindsay, and David Maier, editors, *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 46–57, Portland, Oregon, June 1989.

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

[28] C. Papadimitriou and K. Steiglitz. *"Combinatorial Optimization: Algorithms and Complexity"*. Prentice Hall, 1982.

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

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

[31] H.-J. Schek and M. H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[32] D. Suciu, Fixpoints and bounded fixpoints for complex objects, Technical Report MS-CIS-93-32/L&C 58, University of Pennsylvania, 1993.

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

[34] M. Vardi, The complexity of relational query languages, *Proceedings of ACM SIGACT Symposium on the Theory of Computing*, 1982, pages 137–146.

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

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

[37] D. A. Watt and P. Trinder. Towards a theory of bulk types. Fide Technical Report 91/26, Glasgow University, Glasgow G12 8QQ, Scotland, July 1991.

[38] L. Wong. A conservative property of a nested relational query language. Technical Report MS-CIS-92-59/L&C 48, University of Pennsylvania, Philadelphia, PA 19104, July 1992. Extended abstract to appear in proceedings of PODS'93.

Addendum: Conservative extension in the presence of aggregate functions

In this addendum, we prove that $\mathcal{NRC}(eq_b, \mathbf{N}, \cdot, \dot{-}, +, \sum, cond)$ has the conservative extension property. Here the boolean type *bool* is simulated by $\mathbf{N}$ by representing $true$ using 1 and $false$ using 0. First,

Proposition A *Equality test at all types is definable.*

Proof. We define $= s : s \times s \to bool$ by induction on s.

- $=_b$ is the given equality test at base type b.
- $x =_{s \times t} y := \mathit{if}\ \pi_1\ x =_s \pi_1\ y\ \mathit{then}\ \pi_2\ x =_t \pi_2\ y\ \mathit{else}\ 0$
- $X =_{\{s\}} Y := \mathit{if}\ X \subseteq_s Y\ \mathit{then}\ Y \subseteq_s X\ \mathit{else}\ 0$
- $X \subseteq_s Y := (1 \dot{-} \Sigma\{\mathit{if}\ x \in_s Y\ \mathit{then}\ 0\ \mathit{else}\ 1 \mid x \in X\})$
- $x \in_s Y := \Sigma\{\mathit{if}\ x =_s y\ \mathit{then}\ 1\ \mathit{else}\ 0 \mid y \in Y\}$. □

As mentioned in the main part of the report, existence of a computable linear order is used to prove conservativity of $\mathcal{NRC}(eq_b, \mathbf{N}, \cdot, \dot{-}, +, \sum, cond)$. Hence we need

Proposition B *A linear order at all types is definable.*

Proof. The linear order $\leq_s$ defined in the main report is implemented by induction on s.

- $x \leq_{unit} y := 1$
- $x \leq_{\mathbf{N}} y := (x \dot{-} y) =_{\mathbf{N}} 0$
- $x \leq_{s \times t} y := \mathit{if}\ \pi_1\ x \leq_s \pi_1\ y\ \mathit{then}\ (\mathit{if}\ \pi_1\ x =_s \pi_1\ y\ \mathit{then}\ \pi_2\ x \leq_t \pi_2\ y\ \mathit{else}\ 1)\ \mathit{else}\ 0$
- $X \leq_{\{s\}} Y := \mathit{if}\ X \sqsubseteq^{\flat}_s Y\ \mathit{then}\ (\mathit{if}\ Y \sqsubseteq^{\flat}_s X =_{\mathbf{N}} 0\ \mathit{then}\ 1\ \mathit{else}\ X \leq^{\flat}_s Y)\ \mathit{else}\ 0$, where
- $X \sqsubseteq^{\flat}_s Y := 1 \dot{-} \Sigma\{\mathit{if}\ (\Sigma\{x \leq_s y \mid y \in Y\}) =_{\mathbf{N}} 0\ \mathit{then}\ 1\ \mathit{else}\ 0 \mid x \in X\}$
- $X \leq^{\flat}_s Y := (\Sigma\{\mathit{if}\ x \in_s Y\ \mathit{then}\ 0\ \mathit{else}\ (\mathit{if}\ (\Sigma\{\mathit{if}\ y \in_s X\ \mathit{then}\ 0\ \mathit{else}\ x \leq_s y \mid y \in Y\}) =_{\mathbf{N}} 0\ \mathit{then}\ 1\ \mathit{else}\ 0) \mid x \in X\}) =_{\mathbf{N}} 0$. □

Next we present a rewriting system adapted from Wong [38] for $\mathcal{NRL}(eq_b, \mathbf{N}, \cdot, \dot{-}, +, \sum, cond)$.

1. $(\lambda x.e)(e') \rightsquigarrow e[e'/x]$
2. $\pi_i(e_1, e_2) \rightsquigarrow e_i$

3. $e \rightsquigarrow ()$ if e : *unit* and e is not $()$.

4. *if* 1 *then* e_1 *else* $e_2 \rightsquigarrow e_1$

5. *if* 0 *then* e_1 *else* $e_2 \rightsquigarrow e_2$

6. $\pi_i(\textit{if } e_1 \textit{ then } e_2 \textit{ else } e_3) \rightsquigarrow \textit{if } e_1 \textit{ then } \pi_i\ e_2 \textit{ else } \pi_i\ e_3$

7. $\bigcup\{e \mid x \in \{\}\} \rightsquigarrow \{\}$

8. $\bigcup\{\{\} \mid x \in e\} \rightsquigarrow \{\}$

9. $\bigcup\{e \mid x \in \{e'\}\} \rightsquigarrow e[e'/x]$

10. $\bigcup\{e_1 \mid x \in \bigcup\{e_2 \mid y \in e_3\}\} \rightsquigarrow \bigcup\{\bigcup\{e_1 \mid x \in e_2\} \mid y \in e_3\}$

11. $\bigcup\{e \mid x \in e_1 \cup e_2\} \rightsquigarrow \bigcup\{e \mid x \in e_1\} \cup \bigcup\{e \mid x \in e_2\}$

12. $\bigcup\{e \mid x \in \textit{if } e_1 \textit{ then } e_2 \textit{ else } e_3\} \rightsquigarrow \textit{if } e_1 \textit{ then } \bigcup\{e \mid x \in e_2\} \textit{ else } \bigcup\{e \mid x \in e_3\}$

13. $\Sigma\{e \mid x \in \{\}\} \rightsquigarrow 0$

14. $\Sigma\{e \mid x \in \{e'\}\} \rightsquigarrow e[e'/x]$

15. $\Sigma\{e \mid x \in e_1 \cup e_2\} \rightsquigarrow \Sigma\{e \mid x \in e_1\} + \Sigma\{\textit{if } x \in e_1 \textit{ then } 0 \textit{ else } e \mid x \in e_2\}$

16. $\Sigma\{e \mid x \in \textit{if } e_1 \textit{ then } e_2 \textit{ else } e_3\} \rightsquigarrow \textit{if } e_1 \textit{ then } \Sigma\{e \mid x \in e_2\} \textit{ else } \Sigma\{e \mid x \in e_3\}$

17. $\Sigma\{e \mid x \in \bigcup\{e_1 \mid y \in e_2\}\} \rightsquigarrow \Sigma\{\Sigma\{\textit{if } (\Sigma\{\textit{if } x \in e_1[w/x] \textit{ then } (\textit{if } w = y \textit{ then } 0 \textit{ else } w \leq y) \textit{ else } 0 \mid w \in e_2\}) = 0 \textit{ then } e \textit{ else } 0 \mid x \in e_1\} \mid y \in e_2\}$

This system of rewrite rules preserves meaning of expressions. That is,

Proposition C *If* $e \rightsquigarrow e'$, *then* $e = e'$. □

To prove that $\mathcal{NRL}(eq_b, \mathbf{N}, \cdot, \dot{-}, +, \sum, cond)$ has the conservative extension property, we first show that the normal forms induced by the above rewriting system have set height not exceeding their types and that of their free variables. Observe that any normal form of any expression of type $s \to t$ always looks like $\lambda x.e$ where λ-abstraction does not appear in e. So it suffices to prove

Proposition D *Let* $e : s$ *be an expression of* $\mathcal{NRL}(eq_b, \mathbf{N}, \cdot, \dot{-}, +, \sum, cond)$ *in normal form, where* e *contains no lambda abstraction. Then* $ht(e) \leq \max(\{ht(s)\} \cup \{ht(s) \mid s \textit{ is the type of a free variable occurring in } e\})$.

Proof. The proof proceeds by induction on e. Let $k = \max\{ht(s) \mid s \text{ is the type of a free variable occurring in } e\}$. We present the only case not covered by Wong [38] here. Suppose $e : s$ is $\Sigma\{e_1 \mid x \in e_2\}$. Then e_2 must be a chain of projections on a free variable. Hence $ht(x) < k$. Note that e_1 has type s too. By hypothesis, $ht(e_1) \leq \max\{ht(s), k, ht(x)\} = \max\{ht(s), k\}$. The case holds. □

It remains to check that every expression has a normal form. While rules 15, 16, and 17 seem to increase the "character count" of expressions, it should be remarked that $\Sigma\{e \mid x \in e'\}$ is always rewritten to an expression that decreases in the e' position (see claim III below). This is the key to the proof of the following proposition.

Proposition E *The rewriting system consisting of the above rules is strongly normalizing.*

Proof. A precharacteristic σ is an infinite tuple $(\ldots, \sigma(1), \sigma(0))$ whose components are natural numbers, all but finitely many of which are 0. σ becomes a characteristic if $\sigma(0) > 2$. We write $\sigma[n/i]$ for the σ' such that $\sigma'(i) = n$ but agrees with σ otherwise. We write $\sigma_1 * \sigma_2$ for the σ' such that for each i, $\sigma'(i) = \max(\sigma_1(i), \sigma_2(i))$. An environment φ is a function that assigns characteristics to variables. We write $\varphi[\sigma/x]$ for the φ' such that $\varphi'(x) = \sigma$ but agree with σ otherwise. The size $||e||\varphi$ of expression e in environment φ is defined below as a precharacteristic and is ordered lexicographically.

- $||x||\varphi := \varphi(x)$
- $||()||\varphi := ||c||\varphi := ||\{\}||\varphi := (\ldots, 0, 2)$
- $||\pi_i\ e||\varphi := ||\{e\}||\varphi := \sigma[3 \cdot \sigma(0)/0]$ where $\sigma = ||e||\varphi$.
- $||(e_1, e_2)||\varphi := ||e_1 \cup e_2||\varphi := ||e_1 + e_2||\varphi := ||e_1 \dot{-} e_2||\varphi := ||e_1 \cdot e_2||\varphi := ||e_1 =_b e_2||\varphi := (\sigma_1 * \sigma_2)[\sigma_1(0) \cdot \sigma_2(0)/0]$ where $\sigma_i = ||e_i||\varphi$.
- $||$*if* e_1 *then* e_2 *else* $e_3||\varphi := (\sigma_1 * \sigma_2 * \sigma_3)[\sigma_1(0) \cdot (1 + \sigma_2(0) + \sigma_3(0))/0]$ where $\sigma_i = ||e_i||\varphi$.
- $||\lambda x.e||\varphi := ||e||\varphi[(\ldots, 3)/x]$
- $||(\lambda x.e_1)(e_2)||\varphi := (\sigma_1 * \sigma_2)[\sigma_1(0) \cdot \sigma_2(0)/0]$ where $\sigma_2 = ||e_2||\varphi$ and $\sigma_1 = ||e_1||\varphi[\sigma_2/x]$.
- $||\bigcup\{e_1 \mid x \in e_2\}||\varphi := (\sigma_1 * \sigma_2)[\sigma_1(0)^{\sigma_2(0)}/0]$ where $\sigma_2 = ||e_2||\varphi$ and $\sigma_1 = ||e_1||\varphi[\sigma_2/x]$.
- $||\Sigma\{e_1 \mid x \in e_2\}||\varphi := \sigma[1 + \sigma(n)/n,\ n/0]$ where $\sigma_2 = ||e_2||\varphi$, $\sigma_1 = ||e_1||\varphi[\sigma_2/x]$, $\sigma = \sigma_1 * \sigma_2$, and $n = 3 \cdot \sigma_2(0)$.

It is routine to verify the following claims:

Claim I. $||e[e'/x]||\varphi = ||e||\varphi[||e'||\varphi/x]$

Claim II. Suppose for each x, $\varphi_1(x) \leq \varphi_2(x)$ and $\varphi_1(x)(0) \leq \varphi_2(x)(0)$. Then $||e||\varphi_1 \leq ||e||\varphi_2$ and $(||e||\varphi_1)(0) \leq (||e||\varphi_2)(0)$.

Claim III. Let $e_1 \theta e_2$ be any of $e_1 =_s e_2$, $e_1 \subseteq_s e_2$, $e_1 \in_s e_2$, $e_1 \sqsubseteq^b_s e_2$, $e_1 \leq^b_s e_2$, or $e_1 \leq_s e_2$ as defined earlier. Let $\sigma_i = ||e_i||\varphi$ and $\sigma = ||e_1 \theta e_2||\varphi$. Let $j > (\sigma_1(0) \max \sigma_2(0))$. Then $\sigma(j) = \sigma_1(j) \max \sigma_2(j)$.

Using these claims, a simple case analysis can be performed on the rewrite rules to reveal that whenever $e_1 \rightsquigarrow e_2$, $||e_1||\varphi > ||e_2||\varphi$ for any environment φ. Hence the system is strongly normalizing. □

Putting these propositions together, we have

Theorem F *$\mathcal{NRL}(eq_b, \mathbf{N}, \cdot, \dot{-}, +, \sum, cond)$ has the conservative extension property.* □

By replacing the last rewrite rule with: item $\Sigma\{e \mid x \in \bigcup\{e_1 \mid y \in e_2\}\} \rightsquigarrow \Sigma\{\Sigma\{(e \dot{-} \Sigma\{\Sigma\{x = v \mid v \in e_1[u/y]\} \mid u \in e_2\}) \mid x \in e_1\} \mid y \in e_2\}$, we can also show using a similar technique that

Theorem G *$\mathcal{NRC}(eq_b, cond, +, \dot{-}, \cdot, \sum, \mathbf{Q})$ has the conservative extension property.* □