Invited to a special issue of the *Journal of Computer and System Sciences* selected from ACM Princ. of Database Systems, 1993.

Towards Tractable Algebras for Bags*

Stéphane Grumbach§ — I.N.R.I.A.

Tova Milo¶ — Tel Aviv University

Abstract

Bags, i.e. sets with duplicates, are often used to implement relations in database systems. In this paper, we study the expressive power of algebras for manipulating bags. The algebra we present is a simple extension of the nested relation algebra. Our aim is to investigate how the use of bags in the language extends its expressive power, and increases its complexity. We consider two main issues, namely (i) the impact of the depth of bag nesting on the expressive power, and (ii) the complexity and the expressive power induced by the algebraic operations. We show that the bag algebra is more expressive than the nested relation algebra (at all levels of nesting), and that the difference may be subtle. We establish a hierarchy based on the structure of algebra expressions. This hierarchy is shown to be highly related to the properties of the powerset operator.

*Work supported in part by the Esprit Project BRA FIDE 2.

§I.N.R.I.A., Rocquencourt BP 105, 78153 Le Chesnay, France – Stephane.Grumbach@inria.fr

¶Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel – milo@math.tau.ac.il The research was done while the second author visited INRIA (supported by the Chateaubriand scholarship), and University of Toronto (supported by the Institute for Robotics and Intelligent Systems)

1 Introduction

In the standard approach to database modeling, relations are assumed to be sets, and no duplicates are allowed. For real applications, many systems relax this restriction [MD86, Fis87, HM81, CDV88] and support bags in their data model, often to save the cost of duplicate elimination. Efforts have been made for providing a theoretical framework for such systems. Algebras for manipulating bags were developed by extending the relational algebra [DGK82, KG85, Alb91], and optimization techniques for these algebras were studied [BK90, Mum90, Alb91]. Computational aspects of bags were studied in [BS91]. However, while the expressive power of database languages is of major interest in database research, the expressive power of languages for manipulating bags constitutes a new topic of research.

On the other hand, there has been a wide interest in languages for hierarchical data structures [KV84, TF86]. The complexity and the expressive power of languages for nested relations have been extensively studied [KV88, HS91, PG92, GG92, AFS89, HS89, GV90, GV95]. Collection types have been investigated in [BBN91, BTBW92], in connection with structural recursion. Nested bags, on the other hand had never been addressed.

In this paper, we consider algebraic languages for manipulating nested bags, (i.e. complex objects constructed by tuple and bag constructs). The algebra we present is a simple extension of the nested relation algebra [AB87]. Our aim is to investigate how the use of bags in the language extends its expressive power, and increases its complexity. We consider two main issues, namely (i) the impact of the depth of bag nesting on the expressive power, and (ii) the influence of the algebraic operations on the complexity and the expressive power.

Languages for bags are interesting since they can express natural database query language primitives such as aggregate functions. Moreover, bags can be used to study properties of fundamental database operators, such as *duplicate elimination*.

Operations on bags are sensitive to (i) the presence or absence of elements in the bags (like for sets), and (ii) the number of duplicates of each element in the bags. This leads to a variety of operations greater than for sets. We distinguish for instance between two distinct union operators: *additive union*, which gives the sum of the number of duplicates, and *maximal union*, which gives the maximum of the number of duplicates. Similarly, we consider two powerset operations: the *powerbag*, which outputs a *bag* (potentially with duplicates) of bags, and the *powerset*, which outputs a *set* of bags. For instance, the powerbag of a bag containing n occurrences of a single constant, has cardinality 2^n, while its powerset has cardinality $n+1$. The powerbag, is the natural extension of the classical powerset in the context of bags. Unfortunately, this operation allows the definition of queries with arbitrarily high hyperexponential complexity. For tractability reasons, we chose to include only the powerset operator in the algebra. It offers enough expressive power.

We study the expressive power of the bag algebra, under restrictions on the bag nesting. We first consider the algebra with no nested bags, BALG^1. We prove that (on relational mappings) BALG^1 has more expressive power than the relational algebra. For instance, cardinality comparisons can be expressed. This is due to the counting ability that is offered by bags. In particular, the parity of the cardinality of a relation is definable in presence of an order on the domain. Therefore, BALG^1, unlike the relational algebra, can express queries which are not computable in AC0. Moreover, and again unlike the relational algebra, no 0/1 law holds for queries in BALG^1. Nevertheless, BALG^1 has LOGSPACE data complexity.

We next consider the restriction of the algebra, BALG^2, to types with at most one level of bag nesting. This allows the definition of many aggregate functions, such as count, average, etc. We prove that BALG^2 has more expressive power than RALG^2, the nested relational algebra restricted to sets of sets. The proof technique is very specific, and is based on the pebble game defined in [GV90]. BALG^2 has PSPACE data complexity. This is due to the properties of the powerset operator.

Finally, we consider the algebra, BALG^3, with one more level of bag nesting (bags of bags of bags). We first prove that the hierarchy induced by the number of nested bags collapses to BALG^3 (for relational mappings). BALG^3 allows the expression of all elementary queries, and therefore has the same expressive power as the unlimited nested relational algebra (for nested relational mappings). More generally, for queries over nested bags, all elementary queries are expressible, by allowing in the query intermediate types of bag nesting one level higher than the bag nesting of the input/output.

We also study the complexity of queries in terms of the algebraic operations used. In particular, we establish a hierarchy based on the number of nested powerset operations. This hierarchy is shown to be highly related to the properties of the powerset operator. In particular, we consider the two variants of the powerset operator, and the effect they have on query complexity.

The expressive power of languages for bags has been addressed in some recent papers. The algebra considered here was introduced in [GM93]. Various calculi for bags based on some of the operations of [GM93], were presented in [GMK93], and links with various weak arithmetics were established. The expressive power of languages for bags has been investigated in another setting in [Won93, LW93b, LW93a, LW94].

The paper is organized as follows. In the next section, we briefly present the main definitions. Section 3 is devoted to the algebra. In the following sections, we study the expressive power and the complexity of the algebra, when restricted to bags with one level of nesting (Section 4), two levels of nesting (Section 5), and three or more levels of nesting (Section 6).

2 Preliminaries

In this section, we present the basic framework on types, databases, queries, and complexity measures. We assume the existence of an atomic type U, whose domain is an infinite set of constants. Types are defined recursively using the type U and the tuple and bag constructors. If $T_1, ..., T_k$ are types, then $[T_1, ..., T_k]$ is a *tuple* type, whose domain is the set of tuples whose i^{th} argument is of type T_i. A bag is a (homogeneous) collection of objects that may contain duplicates. If T is a type, then $\{\!\{T\}\!\}$ is a *bag* type, whose domain is the set of bags of objects of type T. We say that an element **n-belongs** to a bag if it belongs to that bag and has exactly n occurrences.

A complex type can be represented by a tree whose nodes denote the bag and tuple constructors. The **bag nesting** of a type T is the maximal number of bag nodes in a path from the root to a leaf.

A **bag database** is a set of named bags. (Following the relational model conventions, we shall sometimes refer to these bags as database relations.) A **bag schema** is an expression $B : T$, where B is a bag name, and T is a bag type. An **instance** of B is a bag of type T.

A **database schema** DB is a finite set of bag schemas with distinct bag names. An **instance** of a database schema DB is a mapping associating with every bag schema in DB an instance of that bag schema.

Queries on bag databases are defined by extending the classical definition of [CH80] for relational queries. A query is a mapping from an input schema $\mathbf{DB} = \{B_1, ..., B_n\}$ to an output schema $\mathbf{S}= \{B_0\}$ with a single bag, mapping instances of the input schema to instances of the output schema. Queries must be computable and generic, i.e. insensitive to isomorphisms on the databases, where isomorphisms for bag databases are defined in the natural way. Two bag databases over the same schema, $(D, B_1, ..., B_n)$ and $(D', B_1', ..., B_n')$, are isomorphic if there is a bijection $h : D \rightarrow D'$, which extends componentwise to tuples such that t k-belongs to B_1 if $h(t)$ k-belongs to B_1', and t k-belongs to B_1' if $h^{-1}(t)$ k-belongs to B_1.

Complexity classes of queries on bags are defined straightforwardly by extending the definition for relational queries. We use as complexity measures the time and space used by a Turing machine to produce a standard encoding of the output database, starting from a standard encoding of the input database. The standard encoding of a bag is similar to that of a set, except that each object is repeated in the encoding as many times as it appears in the bag. The **size** of a database is the size of its standard encoding. This definition may seem surprising. A bag can indeed be encoded more efficiently with the number of occurrences associated to each element. Nevertheless, this does not fit with real situations, where duplicates are explicitly stored, sometimes precisely to avoid the cost of duplicate elimination.

We consider the **data complexity** of queries, i.e. the complexity of the evaluation of a query in terms of the size of the input databases. The data complexity is the complexity of the following recognition problem. Let q be a query, I a database instance, and t some tuple. We use the notation B_i^t to denote a bag containing exactly i occurrences of t and nothing else. In order to decide if t k-belongs to $q(I)$, we consider the language:

$$\{enc(B_k^t)\#enc(I) | B_k^t \subseteq q(I), B_{k+1}^t \not\subseteq q(I)\},$$

where enc is an encoding function[1]. Note that unlike the situation for languages over sets, the size of B_k^t may be not negligible with respect to the size of I. Thus, the data complexity is formally defined with respect to both (i) the size of the input database and (ii) the size of the tuple multiplied by the number of duplicates.

For each Turing complexity class $\mathcal{C}$ there is a corresponding complexity class of queries, which, for simplicity, we also denote by $\mathcal{C}$. The classes considered are based on logarithmic, polynomial, exponential and hyper-exponential functions. For each integer i, we denote by *hyper(i)* the set of hyper-exponential functions with exponentiation of height i. $\mathcal{E}$, denotes the set of *elementary* queries, i.e. queries of hyper-exponential complexity.

A different theoretical paradigm, the relational complexity, was introduced by Abiteboul and Vianu [AV91], to deal with generic database queries. The complexity is relative to a

[1]We assume some standard encoding technique, as for instance the one presented in [AHV94].

new generic model of computation, called the relational machine. Relational complexity applies as well very naturally to queries on bag databases, since they are generic mappings. We haven't investigated this issue in the present paper. Nevertheless, an extension of the relational machines with counters was proposed in [GO93], and it was shown in [GM95], that there are close relationships between bags and counters.

3 An Algebra for Bags

In the following we present an algebra for bag manipulation. This algebra extends the complex object algebra [AB87], in the spirit of the bag algebras presented in [DGK82, Alb91]. We start by presenting the algebraic operators. Next we study their properties, and in particular consider the dependencies of the operations.

We assume that all the operations are typed in a polymorphic way. In particular, for some operations polymorphism means different arities. The restrictions on the type of input assure that the output is a homogeneous bag. For example, bag union ($\cup$) can only be applied on bags of the same type and is undefined otherwise. We use in the following lambda notation. If $e(x)$ is an expression in the algebra with a bag symbol x, then $\lambda x e(x)$, defines a mapping. The type system is obvious and we omit the formal definition.

Let B and B' be bags of type $\{\!\{T\}\!\}$ (unless stated otherwise). Let o be an object of type T, and let $\varphi = \lambda x.e(x)$, $\varphi' = \lambda x.e'(x)$, be two lambda expressions mapping objects of type T to objects of type T', where $e(x), e'(x)$ are algebra expressions containing a variable x. We say that B is a sub-bag of B', denoted $B \subseteq B'$, if whenever o n-belongs to B, then o p-belongs to B' for some $p \geq n$.

Basic bag operations

- *Additive union,* $\biguplus$: $B \biguplus B'$ is a bag of type $\{\!\{T\}\!\}$, such that o n-belongs to $B \biguplus B'$ iff o p-belongs to B and q-belongs to B' and $n = p + q$.

- *Subtraction,* $-$: $B - B'$ is a bag of type $\{\!\{T\}\!\}$, such that o n-belongs to $B - B'$ iff o p-belongs to B and q-belongs to B' and $n = sup(0, p - q)$.

- *Maximal union,* $\bigcup$: $B \bigcup B'$ is a bag of type $\{\!\{T\}\!\}$, such that o n-belongs to $B \bigcup B'$ iff o p-belongs to B and q-belongs to B' and $n = sup(p, q)$.

- *Intersection,* $\bigcap$: $B \bigcap B'$ is a bag of type $\{\!\{T\}\!\}$, such that o n-belongs to $B \bigcap B'$ iff o p-belongs to B and q-belongs to B' and $n = inf(p, q)$.

Constructive operations

- *Tupling,* τ : $\tau(o_1, \ldots, o_k) = [o_1, \ldots, o_k]$, is a k-ary tuple, containing o_i $(i = 0 \ldots k)$ in its i^{th} attribute.

- *Bagging,* β : $\beta(o) = \{\!\{o\}\!\}$ is a bag containing o as a single element, i.e. o 1-belongs to $\beta(o)$.

- *Cartesian product,* $\times$: if B and B' are bags containing tuples of arity k and k' respectively, then $B \times B'$ is a bag containing tuples of arity $k + k'$, such that $o = [a_1, \ldots, a_k, a_{k+1}, \ldots, a_{k+k'}]$ n-belongs to $B \times B'$ iff $o_1 = [a_1, \ldots, a_k]$ p-belongs to B, $o_2 = [a_{k+1}, \ldots, a_{k+k'}]$ q-belongs to B' and $n = pq$.

- *Powerset,* $\mathcal{P}$: $\mathcal{P}(B) = \{b | b \subseteq B\}$ is a bag of type $\{\!\{ \{\!\{ T \}\!\} \}\!\}$ containing **one** occurrence of each sub-bag of B.

Destructive operations

- *Attribute projection,* α_i : $\alpha_i([o_1, ..., o_n]) = o_i$.

- *Bag-destroy,* δ : the operation destroys one level of bag nesting. If B is a bag of type $\{\!\{ \{\!\{ T \}\!\} \}\!\}$, then $\delta(B)$ is a bag of type $\{\!\{ T \}\!\}$, and

$$\delta(\{\!\{ x_1, \ldots, x_n \}\!\}) = x_1 \uplus \ldots \uplus x_n.$$

Filters

- *Restructuring,* MAP: $\mathrm{MAP}_\varphi(B)$ is a bag of type $\{\!\{ T' \}\!\}$, constructed by applying $\varphi = \lambda x.e(x)$ on all the members of B. An element o n-belongs to $\mathrm{MAP}_\varphi(B)$ iff for some l there exist exactly l different elements $o_1, \ldots, o_l$, such that for every $j \in \{1, \ldots, l\}$, $\varphi(o_j) = o$, o_j n_j-belongs to B, and $n = n_1 + n_2 + \ldots + n_l$.

  For instance,
  $$\mathrm{MAP}_{\lambda x\ \beta(x)}(\{\!\{ a, a, b \}\!\}) = \{\!\{ \{\!\{ a \}\!\}, \{\!\{ a \}\!\}, \{\!\{ b \}\!\} \}\!\}.$$
  For simplicity $\mathrm{MAP}_{\lambda x\ \beta(x)}$ will be abbreviated in the following by MAP_β. As another example, $\mathrm{MAP}_{\lambda x\ [\alpha_2(x), \alpha_3(x)]}$ denotes the projection of a tuple type on its second and third arguments. For brevity, we shall denote the map projecting the attributes $i_1, \ldots, i_n$ by $\pi_{i_1, \ldots, i_n}$.

- *Selection,* $\sigma_{\varphi = \varphi'}$: $\sigma_{\varphi = \varphi'}(B)$ is a bag of type $\{\!\{ T \}\!\}$. o n-belongs to $\sigma_{\varphi = \varphi'}(B)$ iff $\varphi(o) = \varphi'(o)$, and o n-belongs to B.

- *Duplicate elimination,* ϵ : $\epsilon(B)$ is a bag containing exactly one occurrence of each object of B. More formally, an object o 1-belongs to $\epsilon(B)$ iff o p-belongs to B for some $p > 0$, and 0-belongs to $\epsilon(B)$ otherwise.

Note that membership and containment tests can be expressed using the algebra operators and equality testing. The nested bag algebra is very similar to the (different variants of the) nested relation algebra. The operations $-, \cap, \cup, \times, \sigma, \mathcal{P}$, when applied to bags where each element occurs at most once, behave exactly as the corresponding relational operations. For MAP, if duplicate elimination is applied after MAP, then the result is the same as for the corresponding nested relation MAP [AB87, BM92, BM93]. Similarly, the *bagging* operator behaves exactly like the corresponding *setting* operation.

All bags can be defined with atomic constants, and the four operations: tupling, τ, bagging, β, additive union, $\uplus$, and Cartesian product, $\times$. These four operations constitute the data definition language [Ull88].

The manipulation of bags is interesting from for the gain in expressive power that is offered, since they allow the definition of several fundamental database primitives.

Clearly, counters can be expressed, by using bags containing a number of occurrences of the same constant. Bags can be used to simulate aggregate functions, such as *sum*, *count* and *average*, for instance.

For example, an integer i can be represented by a bag containing i occurrences of an element, say a. If B is a bag of tuples, then:

$$count(B) = \pi_1(\{\!\{[a]\}\!\} \times B).$$

If B is a bag of bags representing integers, then:

$$sum(B) = \delta(B),$$

and

$$\lfloor average(B) \rfloor = count(\sigma_{\lambda x.(\pi_1(x \times count(B)) \subseteq sum(B))}(\mathcal{P}(sum(B)))).$$

Let BALG denote the algebra for bags containing all the operations defined above. We use the notation BALG_θ, to denote the restriction of BALG without the operation θ.

We next study the dependencies between the different operations. Some dependencies were studied in [Alb91]. In particular, it was shown that $\cap$ and $\cup$ can be expressed by $\uplus$ and $-$. Thus $\text{BALG}_{\cup,\cap}$ has the same expressive power as the full algebra BALG. It was also shown that $\uplus$ cannot be expressed using $\cup, \cap, -$. Not surprisingly, with operations like Cartesian product and restructuring in the language, the redundancy of operations increases, and $\uplus$ becomes expressible by other operations in BALG. It is a consequence of the fact that if B_1, B_2 are k-ary bags, then:

$$B_1 \biguplus B_2 = \pi_{1,\ldots,k}((B_1 \times \{\!\{[a]\}\!\}) \cup (B_2 \times \{\!\{[b]\}\!\})).$$

It is also shown in [Alb91] that $-$ cannot be expressed by $\uplus$, $\cup$ and $\cap$. It can be defined in BALG_- as follows:

$$B_1 - B_2 = \delta(\sigma_{\lambda x.(x \biguplus (B_1 \cap B_2)) = B_1}(\mathcal{P}(B_1))).$$

Remark that the bags constructed in the right hand side of the equation have bag nesting higher than that of the input (use of $\mathcal{P}$). Thus $-$ is defined by increasing the bag nesting.

We next consider the duplicate elimination operation ϵ. Duplicate elimination received attention in the context of object oriented languages. In particular, the independence of the duplicate elimination operation was shown in [BP91]. In contrast, duplicate elimination is redundant in BALG.

Proposition 3.1 $\text{BALG}_\epsilon = \text{BALG}$.

This result follows from the implicit duplicate elimination performed by the powerset operator $\mathcal{P}$. ϵ can be defined as follows: if B is a bag of type $\{\!\{[T_1, ..., T_k]\}\!\}$, then $\epsilon(B) = \delta(\mathcal{P}(B) \cap MAP_\beta(B))$. If B is of type $\{\!\{\{\!\{T\}\!\}\}\!\}$, then $\epsilon(B) = \mathcal{P}(\delta(B)) \cap B$.

Note that in this case again, the nesting of bags is increased in the first equation because of the use of powerset. (The nesting is not increased in the second equation because it is preceded by δ.) For nested bags, it can be avoided using a recursive definition extending the previous one as follows: if the T_i's are the atomic type U, use the previous formula. Otherwise,

$$\epsilon(B) = B \cap (\epsilon(\pi_1(B)) \times \ldots \times \epsilon(\pi_k(B))).$$

We show in the next section that for unnested bags the increase of nesting is essential (for expressing both duplicate elimination and subtraction) .

The set of operations in BALG is therefore not minimal. We included all these operations in BALG for convenience, since the majority of them are provided in classical algebras. The set

$$\{\tau, \beta, \times, \uplus, -, \mathcal{P}, \delta, \pi, \sigma\}$$

is minimal. Tupling, τ, and bagging, β, are just constructors. Cartesian product, $\times$, union, $\cup$ (or similarly $\uplus$), difference, -, projection, π, and selection, σ, constitute a minimal set of operations in the relational algebra, and finally powerset, $\mathcal{P}$, and bag-destroy, δ, change the type nesting of their inputs, which is not done by other operations. Note that other operations are redundant under the assumption that there is no restriction on the type nesting. In Section 4, we see that more operations are needed to get the full power of the algebra if no nesting of bags is allowed.

The operations of powerset and bag-destroy allow the creation of duplicates. The next proposition shows the number of duplicates created by the successive application of $\mathcal{P}$ and δ operations.

Proposition 3.2 Let B be a bag of size n.

- The number of occurrences of each constant in $(\delta\mathcal{P})^i(B)$ is at most exponential in n.
- The number of occurrences of each constant in $(\delta\delta\mathcal{P}\mathcal{P})^i(B)$ is at most $hyper(i+1)(n)$.

Proof : The proof follows from the following claim: if B is a bag containing k constants $a_1, ..., a_k$, with m occurrences of each. Then,

- $\delta(\mathcal{P}(B))$ contains $\frac{m(m+1)^k}{2}$ occurrences of each constant a_i,
- $\delta(\delta(\mathcal{P}(\mathcal{P}(B))))$ contains $2^{(m+1)^k-2}(m+1)^k m$ occurrences of each constant a_i.

We first prove this claim. We analyze the structure of $\mathcal{P}(B)$ and $\mathcal{P}(\mathcal{P}(B))$. $\mathcal{P}(B)$ is a bag of bags, containing $(m+1)^k$ different bags. (The number of bags is due to the fact that each bag can contain between 0 to m occurrences of each constant a_i.) For simple combinatoric reasons, each copy of a constant a_i participates in half of those bags. Thus the total number of copies of a_i in all the bags is $\frac{m(m+1)^k}{2}$, and this is exactly what we get when applying δ. It follows that the number of occurrences of a_i in $\delta(\mathcal{P}(B))$ is $\frac{m(m+1)^k}{2}$.

Since all the bags in $\mathcal{P}(B)$ are different from each other, the number of bags (of bags) in $\mathcal{P}(\mathcal{P}(B))$ is $2^{(m+1)^k}$. Here again, every bag of $\mathcal{P}(B)$ participates in half of the nested bags of $\mathcal{P}(\mathcal{P}(B))$. Thus the total number of occurrences of each unnested bag is $2^{(m+1)^k-1}$. It follows that the number of occurrences of each constant a_i in $\mathcal{P}(\mathcal{P}(B))$ is $2^{(m+1)^k-1} \times \frac{m(m+1)^k}{2} = 2^{(m+1)^k-2}(m+1)^k m$, and this is exactly what we get when applying two successive δ's. This concludes the proof of the claim.

We next use the above result, and prove the proposition by induction on i.
Basis: Let $i = 1$. The number of different constants in B is bounded by n. The maximal number of occurrences of each constant is also bounded by n. Thus, from the above claim, the number of occurrences of each constant in $\delta(\mathcal{P}(B))$ is at most $\frac{n(n+1)^n}{2}$, i.e. at most exponential in n. The number of occurrences of each constant in $\delta(\delta(\mathcal{P}(\mathcal{P}(B))))$ is at most $2^{(n+1)^n-2}(n+1)^n n$, i.e. at most $hyper(2)(n)$.
Induction: Assume that the number of occurrences of each constant in $(\delta\mathcal{P})^i(B)$ is at most $2^{P(n)}$ for some polynomial P, and that the number of occurrences of each constant in $(\delta\delta\mathcal{P}\mathcal{P})^i(B)$ is at most $hyper(i+1)(n)$. The number of different constants is the same as in B, thus is at most n. From the above claim, the number of occurrences of each constant in $(\delta\mathcal{P})^{i+1}(B)$ is at most $\frac{2^{P(n)}(2^{P(n)}+1)^n}{2}$ i.e. at most exponential in n. The number of occurrences of each constant in $(\delta\delta\mathcal{P}\mathcal{P})^{i+1}(B)$ is at most $2^{((hyper(i+1)+1)^n-2}((hyper(i+1)+1)^n(hyper(i+1))$ i.e. at most $hyper(i+2)(n)$. $\Box$

It follows that every two consecutive applications of powerset lead to an exponential explosion of duplicates, while applying only one powerset leads at the first step to an exponential explosion, but further applications lead only to a polynomial explosion. This difference is a key argument in the main results of the following sections.

Like in the relational algebra, the operations satisfy some algebraic properties, such as associativity ($\uplus$, $\cup$, $\cap$, $\times$), commutativity ($\uplus$, $\cup$, $\cap$), etc. These properties can be used to define rewriting rules, to optimize queries over bags, in the same spirit as optimization of queries over sets, by pushing down selections for instance [BK90, Mum90, Alb91]. Nevertheless, it was shown in [CV93], that classical techniques to optimize conjunctive queries over sets do not carry over under a bag semantics. It was shown in particular that the containment of conjunctive queries over bags is Π_2^p-hard, while it is NP-complete over sets. On the other hand, the equivalence of conjunctive queries over bags has the same complexity as graph isomorphism, while it is NP-complete for queries over sets.

Queries over bags can also be optimized by representing each object in association with the number of its occurrences, instead of storing explicitly duplicates. Some simple arithmetic is then done on the number attached to each object as a result of algebraic operations. It is not clear though how this can be used in practice. Indeed, at the physical level, duplicates, and therefore bags, result often precisely from an optimization principle based on avoiding the costly duplicate elimination (after projections for instance).

In the following sections, we study the expressive power of BALG restricted to types with bounded nesting of bag constructors. We denote the algebra when restricted to bag nesting of depth k, BALG^k. We start by investigating the properties of BALG^1.

4 Unnested Bags, BALG^1

The operations of BALG are defined for bags of any type. BALG^1 is the restriction of the algebra to unnested bag types, i.e. U^k and $\{\!\{U^k\}\!\}$, for every k. So in particular, BALG^1 does not contain $\mathcal{P}$ nor δ. Restricting the type system to unnested bags, limits the expressive power. For example, the duplicate elimination ϵ which is redundant in BALG (Proposition 3.1), is not redundant in BALG^1. The same holds for the subtraction.

Proposition 4.1

- $\mathrm{BALG}^1_\epsilon \subset \mathrm{BALG}^1$.
- $\mathrm{BALG}^1_- \subset \mathrm{BALG}^1$.

Before proving Proposition 4.1, we illustrate a technique used in the proof, to count the number of occurrences of each tuple in the result of a query, depending on the number of occurrences of each tuple in the input.

Let B be a binary bag containing n occurrences of $[a,b]$, and m occurrences of $[b,a]$, and nothing else. Consider the query: $Q(B) = \pi_{1,4}(\sigma_{\alpha_2=\alpha_3}(B \times B))$, where $\sigma_{\alpha_2=\alpha_3}$ is a shorthand for $\sigma_{\lambda x.\alpha_2(x)=\lambda x.\alpha_3(x)}$. The number of occurrences of each tuple is given by the following functions:

tuple	B	$Q(B)$
ab	n	0
ba	m	0
aa	0	nm
bb	0	nm

tuple	$B \times B$	$\sigma_{\alpha_2=\alpha_3}(B \times B)$
$abab$	n^2	0
$baba$	m^2	0
$baab$	nm	nm
$abba$	nm	nm

Proof of Proposition 4.1 To prove the first statement we show that there is no expression $e(B)$ using the operations $\uplus$, $\cup$, $\cap$, $-$, τ, β, $\times$, MAP and σ, such that $e(B)$ is equivalent to $\epsilon(B)$ for any bag B. The proof is based on the following claim.

Claim : Let e be a BALG^1_ϵ expression. Let a be a constant that does not appear in e. For every tuple t there exists some number N_t, and a polynomial $P_t(n) = k_0 + k_1 n + k_2 n^2 + \ldots + k_m n^m$ such that for every bag B_n of size $n > N_t$ containing n occurrences of a single tuple $[a]$, the number of t's in $e(B_n)$ is $P_t(n)$. Moreover, if the constant a appears in t, then $k_0 = 0$.

Since $\cup$ and $\cap$ can be expressed by other operations, we consider only expressions constructed using $\uplus, -, \tau, \beta, \times, \alpha_i$, MAP, and σ. The proof of the claim is by induction on the size of e.

Let e be an expression whose result is a k-ary bag, and let t be a tuple. Clearly, if t is not a k-ary tuple, then t cannot belong to the result of e, and $P_t = 0$, $N_t = 0$. Similarly, if t contains constants that neither appear in e nor in the input bag, t cannot belong to the

result of e, and $P_t = 0$, $N_t = 0$. Thus we only have to consider k-ary tuples constructed from the constant a and from the constants in e. Let $t_1, \ldots, t_n$ be all theses possible tuples.
Basis: If e has no operations, (i.e. $e(B) = B$), then we only have to consider the tuple $[a]$, and $P_{[a]}(n) = n$, $N_{[a]} = 0$. If o is a constant or a tuple of constants, and $e = \tau(o)$, $e = \beta(o)$, or $e = \alpha_i(o)$, then $P_{[o]}(n) = 1$, $N_{[o]} = 0$, $P_{[a]}(n) = 0$, $N_{[a]} = 0$.
Induction: Assume that the claim holds for expressions with less than i operators, and let e be an expression with $i+1$ operators.

- If $e = e_1 \uplus e_2$, then $P_{t_i} = P^1_{t_i} + P^2_{t_i}$, and $N_{t_i} = max(N^1_{t_i}, N^2_{t_i})$, where $P^j_{t_i}$, $N^j_{t_i}$ are the polynomial and number of t_i in e_j (j=1,2) respectively.

- If $e = e_1 - e_2$, then if $\lim_{n\to\infty}(P^1_{t_i}(n) - P^2_{t_i}(n)) > 0$ then $P_{t_i} = P^1_{t_i} - P^2_{t_i}$ and $N_{t_i} = max(N^1_{t_i}, N^2_{t_i})$. Otherwise, $P_{t_i} = 0$ and N_{t_i} is the largest n such that $P^1_{t_i}(n) - P^2_{t_i}(n)$ is positive, or 0 if there is no such n.

- If $e = e_1 \times e_2$, then for every t_i, $P_{t_i} = P^1_{t^1_i} \times P^2_{t^2_i}$, and $N_{t_i} = max(N^1_{t^1_i}, N^2_{t^2_i})$, where $t = [t^1_i, t^2_i]$, and $P^j_{t^j_i}$, $N^j_{t^j_i}$ are the polynomial and number of t^j_i in e_j (j=1,2) respectively. Note that if t^1_i or t^2_i contain a, then the constant factor in $P^1_{t^1_i} \times P^2_{t^2_i}$ is 0.

- If $e = \mathrm{MAP}_\varphi(e_1)$, where φ is a λ expression, and e_1 is a k'-ary expression, then $P_{t_i} = \Sigma_t P_t$, (where t ranges over all the k'-ary tuples constructed from a and from the constants in e_1 such that $\varphi(t) = t_i$), and $N_{t_i} = max(N_t)$. Note that the number of relevant tuples depends only on the number of constants appearing in e_1, and on the arity of e_1. Thus $\Sigma_t P_t$ is a polynomial. Also note that if t_i contains a, then so does all the t's. This is because a does not appear in φ. Thus the constant factor in all the P_t's is 0, and is therefore also 0 in P_{t_i}.

- If $e = \sigma_{test}(e_1)$, then if t_i satisfies $test$ then $P_{t_i} = P^1_{t_i}$, and $N_{t_i} = N^1_{t_i}$, (where $P^1_{t_i}$, $N^1_{t_i}$ are the polynomial and number of t_i in e_1). Else $P_{t_i} = 0$, and $N_{t_i} = 0$.

This proves the claim. We next use the claim to prove the first statement of the proposition. Consider the polynomial of the tuple $t = [a]$. In this polynomial the constant factor is 0. Since for every such polynomial P there are integers n greater than N such that $P(n) \neq 1$, the expression e does not perform duplicate elimination for the bag B_n of size $n > N$ containing n occurrences of $[a]$. Thus, $e \neq \epsilon$.

The second part of Proposition 4.1 follows from Propositions 4.2 and 4.3 below. □

As in the classical relational case, we are aiming for characterizations of expressiveness of BALG^1 in terms of complexity classes of queries. In particular, we compare the expressive power of the bag algebra to that of the relational algebra. In the following we denote the relational algebra by RALG. As for bags, we use the notation RALG_θ to denote RALG, where operation θ is removed. When comparing BALG^1 and RALG, we consider queries over the same input. We restrict our attention to BALG^1 queries mapping set inputs to set outputs.

Proposition 4.2 The algebra BALG^1_- has the same expressive power as RALG_- over sets.

Proof : Clearly, every RALG_- query can be expressed in BALG^1_-, by adding a duplicate elimination operation after each operator.

For the other direction we show that for every BALG^1_- expression Q, there exists a RALG_- expression Q', such that for every element a, and every database instance DB, $a \in Q(DB)$ iff $a \in Q'(DB')$, where DB' is obtained from DB by applying duplicate elimination on each database relation. In particular, this implies that if DB is a relational database, i.e. each relation is a set, and $Q(DB)$ is a set too, then $Q(DB) = Q'(DB)$. (Note that in general $Q(DB)$ may contain several occurrences of a while $Q'(DB')$ contains only one.)

Q' is constructed from Q by replacing BALG^1_- operations with corresponding RALG_- operations, or by simply omitting operations. $\uplus$ is replaced by the relational union. $\cup$, $\cap$, $\times$, and σ are replaced by relational union, intersection, Cartesian product, and selection respectively. τ is the standard tuple construction. β is replaced by set construction. Observe that in BALG^1, the φ of MAP_φ operates on tuples. Thus MAP can only project certain attributes, and/or add some constant attributes. Thus it can be simulated by π and $\times$ (π is used for projecting attributes, and Cartesian product with a set containing a constant is used for adding an attribute containing that constant). Finally, ϵ is simply omitted.

A simple induction on the size of Q is used to show that for every element a, and every database instance DB, $a \in Q(DB)$ iff $a \in Q'(DB')$. The proof is trivial, and therefore omitted. $\Box$

It turns out that the equivalence no longer holds when the difference operator is used. We next present an example illustrating the power of the bag difference.

Example 4.1 Consider a directed graph whose edges are recorded in a binary relation G. The following query expresses the fact that the in-degree of a node a is bigger than its out-degree.

$$(\pi_2(\sigma_{2=a}G)) - (\pi_1(\sigma_{1=a}G)) \neq \emptyset,$$

where $\sigma_{i=a}$, is a shorthand for $\sigma_{\lambda x.\alpha_i(x)=a}$ for $i = 1, 2$.

This example shows the power of the language, since the above query is not even expressible in the infinitary logic $\mathcal{L}^\omega_{\infty,\omega}$ [KV92]. $\mathcal{L}^\omega_{\infty,\omega}$ is the extension of first-order logic to infinite formulas with infinite conjunctions and disjunctions but a finite number of variables. Infinitary logic subsumes various kinds of fixpoint logics, but has weak counting ability. The bags give a counting power. Indeed, counting quantifiers [IL90] of the form "there exists at least i x's", $\exists_i x$, Härtig (Rescher) quantifiers of the form "there exists equally many (less) x's satisfying property P and (than) property Q", are all definable in BALG^1. It suffices to compute the Cartesian product of $\{\!\{[a]\}\!\}$ with the set to count, to project on the first attribute, and then to compare using the subtraction operator $-$.

The following proposition therefore follows from Example 4.1.

Proposition 4.3 The algebra BALG^1 has more expressive power than RALG.

Note that the asymptotic probabilities of properties defined in BALG^1 differ from the asymptotic probabilities of properties defined in RALG on unnested inputs. For simplicity, consider databases which are graphs, that is binary relations. The probability, $\mu_n(P)$, that a (boolean) property P defined in RALG holds over graphs with n vertices is the ratio of the number of graphs with n vertices satisfying P, on the number of graphs with n vertices:

$$\mu_n(P) = \frac{|\{G \mid G = (V, E), |V| = n, G \models P\}|}{2^{n^2}}.$$

The asymptotic probability of P is the limit of this ratio (if it exists) when n goes to ∞. Boolean expressions in RALG containing no constants admit a 0/1 law (that is the asymptotic probability exists and can only be 0 or 1), while BALG^1 doesn't enjoy such a regularity. Indeed, the query introduced in the next example has asymptotic probability $\frac{1}{2}$, for instance.

Example 4.2 Consider a schema over two monadic relation symbols R and S. The next query expresses the fact that the cardinality of R is bigger than the cardinality of S:

$$(\pi_1(R \times R) - \pi_1(R \times S)) \neq \emptyset.$$

The asymptotic probability of the above query is $\frac{1}{2}$. The result follows from [FGT93], where it is shown that first-order sentences with limited Rescher's quantifiers (expressing cardinality comparison) have asymptotic probability 0, $\frac{1}{2}$, or 1. The proof involves classical methods from complex analysis. The intuitive idea in this case goes as follows. The probability that R and S have the same size goes to 0 when the number of different constants and the number of occurrences of the same constant goes to ∞. Now, for each input (R, S) with relations of different sizes, there is probability $\frac{1}{2}$ that it satisfies the property. Indeed, either (R, S) or (S, R) satisfies the property. For more details on the asymptotic probabilities of queries expressing counting properties, see [GT95, FGT93].

BALG^1 also differs from RALG for its data complexity. Indeed BALG^1 doesn't enjoy an AC0 data complexity upper-bound. AC0 [FSS84] is the class of problems that can be solved on boolean circuits, with arbitrary fan-in gates, of constant size and polynomially many processors. The ACO upper-bound offers potential for efficient parallel evaluation. RALG enjoys an AC0 upper-bound [AHV94]. It is well known that there are simple functions that are not computable in AC0, such as the function MAJORITY which compares cardinalities [FSS84]. It follows from Example 4.2, that BALG^1 is not in AC0. The complexity of BALG^1 is nevertheless not too dramatic as shown in the next proposition.

Theorem 4.4 $\text{BALG}^1 \subset$ LOGSPACE.

Proof : We first show that BALG^1 is included in LOGSPACE. The proof goes along the same lines as for the relational algebra, RALG (for a complete proof see [AHV94], page 430). The logspace upper bound is obtained as follows. When computing a RALG query, tuples are not copied on the work tape, but instead, their addresses in the input tape are used on the work tape. This allows the whole computation to be done in logarithmic space. The main difference in presence of bags comes from the duplicates. We next show that the classical proof carries over for BALG^1, when information on the number of duplicates is added to the addresses of the tuples on the work tape. With each tuple, we associate the number of its occurrences. That is when the address of a tuple is written on the work tape, it is followed by a number (number of occurrences). From the definition of the operators of BALG^1, it follows (see proof of Proposition 4.1) that the maximal number of occurrences of each tuple at any step of the evaluation of a query is bounded by some polynomial in the number of duplicates in the input, and so in the size of the input. Thus, these numbers can be encoded in space logarithmic in the size of the input. The fact that the inclusion is strict follows from the next proposition. $\Box$

Consider bags containing occurrences of a single constant. The query *bag-even* distinguishes the parity of the number of duplicates in a bag. More precisely, *bag-even*$(B) = B$ if the number of duplicates in B is even, and $\emptyset$ otherwise.

Proposition 4.5 The query *bag-even* is not expressible in BALG^1.

Proof : The proof is very similar to that of Proposition 4.1. It is based on the fact that for every BALG^1 expression e, and every tuple t, there exists some number N_t, and a polynomial $P_t(n) = k_0 + k_1 n + k_2 n^2 + \ldots + k_m n^m$ such that for every bag B_n of size $n > N_t$ containing n occurrences of $[a]$, the number of t's in $e(B_n)$ is $P_t(n)$.

The proof of this fact is similar to that of the claim in the proof of Proposition 4.1 with an additional step in the induction for handling the duplicate elimination operator ϵ.

- if $e = \epsilon(e_1)$, $N_{t_i} = N^1_{t_i}$, and $P_{t_i} = 1$ if $P^1_{t^1_i} \neq 0$, and equals 0 otherwise. (where $P^1_{t_i}$, $N^1_{t_i}$ are respectively the polynomial and number of t_i in e_1).

Note that every such polynomial, P_t, cannot get infinitely many times the values 0 and 1, and eventually (for large enough n) it is monotone (either decreasing or increasing). Thus, there is an integer $N \geq N_t$ such that the expression e does not compute the query *bag-even* for a bag B_n of size n. Thus, $e \neq$ *bag-even*. $\Box$

The previous proposition implies that the parity of the cardinality of a bag cannot be defined in BALG^1. Moreover the proof also shows that any boolean expression $e(B)$ in BALG^1, where B is a bag variable ranging over bags over a single constant, is satisfied by a finite or co-finite number of bags, as noted in [LW93b].

On the other hand, the *parity* of the cardinality of a relation (bag with no duplicates) becomes definable in BALG^1 in the presence of an *order* on the domain. This was first shown in [LW93a] for a language equivalent to BALG^1. The following boolean expression states that the parity of the cardinality of relation R is even:

$$\sigma_{\lambda x \ (\mathrm{MAP}_{[a]}(\sigma_{\lambda y \ (y \leq x)} R) = \mathrm{MAP}_{[a]}(\sigma_{\lambda y \ (x < y)} R))}(R) \neq \emptyset.$$

The previous expressions states the existence of an x such that the number of elements smaller than or equal to x equals the number of elements strictly bigger than x. (The counting is simulated using bags containing $[a]$ tuples, one for each element). It is clear that the existence of such an element in R guarantees parity of the cardinality of R.

It should be recalled that the parity of the cardinality of a relation is not first-order definable even in the presence of an order relation. This is easily proved using Ehrenfeucht-Fraïssé [Ehr61, Fra54] games. This shows the difference between BALG^1 and RALG in presence of an order on the domain.

It has been shown in [LW94], that the parity of the cardinality of a relation is not definable in BALG^1 in general, that is without assuming an order relation over the domain. The proof of [LW94], is done in a more general setting than BALG^1. It was also proved in the same paper, that the *transitive closure* of a binary relation is not definable in BALG^1. Libkin and Wong introduced a new technique to prove this result. In the case of first-order logic, this result can be proved using Ehrenfeucht-Fraïssé games. This proof technique, as well as techniques based on 0/1 laws do not work anymore in the context of BALG^1, as follows from Example 4.2.

5 Nested Bags, BALG^2

In the previous section, we considered a restricted version of the algebra, where only unnested bags were manipulated. We next consider BALG^2, one stage higher in the nesting hierarchy. Languages for bags allowing nested bags are interesting since they allow the definition of many database language primitives, such as counting, already expressible in BALG^1, and moreover aggregate functions, such as *sum*, *count* and *average*, as shown in Section 3.

As we did before, we study the intrinsic expressive power of BALG^2, and we then compare it to algebras for manipulating (nested) sets. The next theorem presents an upper bound on the complexity of BALG^2.

Theorem 5.1 BALG^2 is in PSPACE.

Proof : The theorem is proved by showing that the size of the unnested bags used in the computation of a BALG^2 query is bounded by an exponential in the size of the input, and that they can be encoded in polynomial space. First observe that as in the nested relational algebra, the number of different tuples used in the computation is at most polynomial in the size of the input. (It depends on the number of different constants in the input, and the arity of the bags used in the computation). To prove the theorem we also use the following claim.

Claim : For every BALG^2 expression $e(B_1, \ldots, B_m)$, there exists a polynomial P such that,
(i) If $e(B_1, \ldots, B_m)$ is an unnested bag, then it contains at most $2^{P(n)}$ occurrences of each different tuple, (were n is the maximal size of $B_1, \ldots, B_m$).
(ii) If $e(B_1, \ldots, B_m)$ is a nested bag, then it contains at most $2^{P(n)}$ occurrences of unnested bags, were every such bag contains at most $2^{P(n)}$ occurrences of each different tuple.

The two claims are proved simultaneously by induction on the number of operators in e.
basis: The claim clearly holds when e contains no operators, i.e. when $e = B_i$ for some $i = 1 \ldots m$.
Induction: Assume that the claim holds for expressions with less than i operators. Let e_1, e_2

be expressions with less than i operators, and let P_1, P_2 be their corresponding polynomials. If $e = e_1 \uplus e_2$, and the outputs of e_1, e_2 are unnested bags, then e contains at most $2^{P_1(n)} + 2^{P_2(n)} \leq 2^{P_1(n)+P_2(n)}$ occurrences of each tuple. If the outputs of e_1, e_2 are nested bags, then e contains at most $2^{P_1(n)} + 2^{P_2(n)} \leq 2^{P_1(n)+P_2(n)}$ bags, where in each bag the number of occurrences of every tuple is bounded either by $2^{P_1(n)}$ or $2^{P_2(n)}$. Thus, it is clearly less than $2^{P_1(n)+P_2(n)}$. Similar arguments hold for $-, \bigcup, \bigcap, \times, \sigma$, MAP, τ, and β.
We next consider the case where $e = \mathcal{P}(e_1)$. Since BALG^2 supports only two levels of nesting, e_1 must be an unnested bag. Thus it contains at most $P_t(n)$ different tuples for some polynomial P_t, and at most $2^{P_1(n)}$ occurrences of each such tuple. Thus the number of different bags constructed from these tuples is at most $(2^{P_1(n)} + 1)^{P_t(n)} < 2^{2P_1(n)P_t(n)}$. (The number of bags is due to the fact that each bag can contain between 0 to $2^{P_1(n)}$ occurrences of each tuple.) The number of occurrences of tuples in each of these bags is bounded by their number in e_1, thus clearly smaller than $2^{2P_1(n)P_t(n)}$.
Finally, consider $e = \delta(e_1)$. To apply the bag-destroy operator, the output of e_1 must be a nested bag. The number of occurrences of a tuple t in $\delta(e_1)$ is bounded by the maximal number of occurrences of t in a subbag of e_1 times the number of the subbags. Thus is bounded by $2^{2P_1(n)}$. This concludes the proof of the claim.

We use the above claim to prove Theorem 5.1, and show that the computation can be done in polynomial space. The technique is classical [AHV94, HS91], and we present here only the points that are specific to our case. We consider a Turing machine whose input tape has the following structure.

$$\boxed{[enc(t_1), ..., enc(t_n)]\#...\#[enc(t_1), ..., enc(t_n)]\#\#enc(db)}$$

The function enc encodes the atomic constants into words over the alphabet $\{0, 1\}$, and recursively encodes complex objects. db is the input database, and $[t_1, ..., t_n]$ is a tuple (of either atomic constants, or bags of tuples of atomic constants). The TM answers positively if this tuple belongs to the output of the query on the input db, and with exactly the same number of duplicates as appearing in the input tape. The critical step is to show that the bags can be enumerated within polynomial space. The unnested bags can be encoded using a technique similar to the one used for unnested sets [HS91], where with each element we also associate the number of its occurrences. For nested bags, we enumerate their elements. This can be done as follows. From an order on the atomic constants, we can derive a (lexicographic) order on tuples, and then on sets and bags of tuples by taking again the lexicographic order over the sequences obtained by ordering the elements of the set from the biggest to the smallest ones, and by keeping duplicates of an element in a row.

The encoding differs from that of nested sets by the fact that with each element, the number of its occurrences is associated. Since both (i) the number of occurrences of tuples of unnested bags in a nested bag, and (ii) the number of flat tuples in an unnested bag is at most exponential in the size of the input, they can be enumerated in space polynomial in the size of the input. Thus, the whole encoding can be done in space polynomial in the size of the input. The rest of the proof is carried out as in the case of nested sets. □

We next consider the relationship between the nested bag algebra and the nested relation algebra. The two algebras are very close. The operations of the nested relation algebra are similar to those of the bag algebra, but operate only on (nested) sets. We denote the nested relation algebra when restricted to set nesting of depth k, RALG^k [HS91, GV95]. We compare RALG^2 and BALG^2 restricted to queries over (nested) sets. It was shown in [HS91] that RALG^2 is in PSPACE. It follows from Theorem 5.1 that BALG^2, restricted to queries over (nested) sets, is in PSPACE too. Clearly $\mathrm{RALG}^2 \subseteq \mathrm{BALG}^2$. In the next theorem we see that the inclusion is proper.

Theorem 5.2 $\mathrm{RALG}^2 \subset \mathrm{BALG}^2$.

To prove the theorem, we consider the following query Φ on graphs. It checks whether the in-degree of a given node is bigger than its out-degree. This query was shown in Example 4.1 to be expressible in BALG^1. Similarly, it can be expressed in BALG^2 for graphs whose nodes are sets. We prove the theorem by showing that Φ is not expressible in RALG^2 for such graphs.

Before proving the above theorem, note that we use a query with nested input. The same result restricted to queries on flat (non-nested) inputs only, would imply a separation between the polynomial time hierarchy, PHIER, and polynomial space, PSPACE. (Indeed, RALG^2 restricted to flat inputs is equivalent to second-order logic, which has been shown to characterize PHIER [Fag75].)

The proof is based on the pebble games introduced in [GV90] to characterize the expressive power of calculi for nested relations. It was shown in [AB87] that the relational nested algebra, RALG^2, and the calculus with quantification over sets of tuples of atoms, CALC^1, were equivalent. We consider a game which characterizes exactly the expressive power of CALC^1 and therefore this game technique can be used to prove non definability in RALG^2.

We first briefly recall the definition of CALC^1, an extension of the relational calculus to complex objects as defined in [HS91]. CALC^1 is a typed calculus with the constructible types tuple [ ] and set { }. The calculus uses typed variables and typed relation symbols. There is a function $.i$ which associates to a tuple its i^{th} component. There are also three typed logical predicates: membership ($\in_T$), set containment ($\subseteq_T$), and equality ($=_T$). Usual type compatibility conditions must be obeyed in well-formed formulas.

The semantics is defined using the notion of *active domain*, meaning that the quantifiers in the formulas range over sets and tuples constructed from atomic constants in the input. In the evaluation of a formula φ on an input A, each quantified variable of type T ranges over the domain $dom(T, A)$, i.e. the set of objects of type T constructed using the atomic constants occurring in A. Let $\mathcal{T}$ be the set of types occurring in φ. In order to evaluate φ, an interpretation for the logical predicate symbols $\in, \subseteq, =$ involving types in $\mathcal{T}$ is also needed. The structure A can be extended in a natural manner to a structure interpreting the logical predicates involving types in $\mathcal{T}$. The extended interpretation is denoted by $Comp(A, \mathcal{T})$, and called the *completion* of A with respect to $\mathcal{T}$. The non-logical predicates are interpreted as in A, and the logical predicates involving types in $\mathcal{T}$ are interpreted in the standard fashion over objects in $\{dom(T, A) | T \in \mathcal{T}\}$. By definition, $A \models \varphi$ iff $Comp(A, \mathcal{T}) \models \varphi$, where $\mathcal{T}$ is the set of types of φ.

It has been shown in [GV90] that a direct extension of the Ehrenfeucht-Fraïssé games [Ehr61, Fra54] for structures with complex objects cannot be used to characterize CALC^1

sentences, since a CALC_1 sentence may use types which do not exist in the structures on which the game is played, and even if the structure and the sentence use the same types, the sentence may refer to objects which do not appear in the structures. Nevertheless, the game can be modified to characterize the calculus for complex objects. We next briefly describe the modified version of the game. The definition appeared in [GV90].

Let A, A' be two nested inputs, and let D, D' be the sets of constants used in A, A' respectively. The *game* with k moves with respect to a set of types $\mathcal{T}$, associated with A and A' is played by two players, the *spoiler* and the *duplicator*, making k moves each. The spoiler starts by picking an object of some type T in $\mathcal{T}$, say in $\mid Comp(A, \mathcal{T}) \mid$, the domain of $Comp(A, \mathcal{T})$. The duplicator answers by picking an object of type T in the opposite structure, $\mid Comp(A', \mathcal{T}) \mid$, the domain of $Comp(A', \mathcal{T})$. This is repeated k times. At each move, the spoiler has the choice of the structure, and the duplicator must respond in the opposite structure.

As in the classical Ehrenfeucht-Fraïssé game, the objective of the game for the duplicator is to choose isomorphic substructures in the two structures. The notions of substructure and isomorphism are extensions from the classical case. An isomorphism f from A to A' is a bijection from $\mid A \mid$ to $\mid A' \mid$ such that for each $a \in \mid A \mid$, a and $f(a)$ have the same type, and the extension F of f on $\mid A \mid \cup \{a.i / a \in \mid A \mid, a \text{ is of tuple type}\}$ defined by $F(a.i) = f(a).i$ preserves the logical and non-logical relations in A and A'. Now, the *substructure* of a structure A generated by a set of objects $C \subseteq \mid A \mid$, denoted A/C, is the restriction of the logical and non-logical relations of A to $C \cup \{c.i / c \in C, c \text{ of tuple type}\}$.

Let a_i (a'_i) be the i^{th} object picked in $Comp(A, \mathcal{T})$ ($Comp(A', \mathcal{T})$). The duplicator wins the round $\{(a_1, a'_1), ..., (a_k, a'_k)\}$ iff the mapping $a_i \to a'_i$ is an isomorphism of the substructures of $Comp(A, \mathcal{T})$ and $Comp(A', \mathcal{T})$ generated by $\{a_1, ..., a_k\}$ and $\{a'_1, ..., a'_k\}$, respectively. The duplicator *wins the game with k moves* associated with A and A' if he has a winning strategy, i.e. the duplicator can always win any game with k moves on A and A', no matter how the spoiler plays. This is denoted by $A \equiv_{k,\mathcal{T}} A'$. Note that the relation $\equiv_{k,\mathcal{T}}$ is an equivalence relation on structures.

It was shown in [GV90] that $A \equiv_{k,\mathcal{T}} A'$ holds iff for every sentence φ in CALC^1 with k variables and types in $\mathcal{T}$, $A \models \varphi$ iff $A' \models \varphi$. More generally, we have the following.

Theorem 5.3 Let Φ be a boolean query on inputs over a schema σ consisting of nested relations. Let $\mathcal{T}$ be a set of types of the form U, $[U, ..., U]$, or $\{[U, ..., U]\}$, and containing the types of the attributes of the relations in σ. Then, the following are equivalent.

1. No expression in RALG^2 over sets of tuples of objects whose types are in $\mathcal{T}$ expresses Φ.

2. For each k, there are no sentences in CALC^1 with types in $\mathcal{T}$, and of quantifier depth k, expressing Φ.

3. For each k, there exist two databases over σ, $\mathcal{A}_k$, and $\mathcal{A}'_k$, which differ with respect to Φ, and such that the duplicator has a winning strategy in the game with k moves with respect to $\mathcal{T}$.

The equivalence of 1. and 2. follows from [AB87]. The equivalence of 2. and 3. follows from [GV90]. The integer k is the same in statements 2. and 3.

The proof of Theorem 5.2 is based on the following lemma.

Lemma 5.4 Consider the class of graphs whose nodes are sets of atomic constants. The property that the in-degree of a specific node is bigger than its out-degree is not definable in CALC^1 (RALG^2).

Proof of Lemma 5.4 : We prove that for every k and every set of types $\mathcal{T}$, there exist two directed graphs $G_{k,\mathcal{T}}$ and $G'_{k,\mathcal{T}}$, such that in $G_{k,\mathcal{T}}$, a node α has the same in- and out-degree, while in $G'_{k,\mathcal{T}}$, α has an in-degree bigger than its out-degree, but the duplicator has a winning strategy for the game with k moves with respect to $\mathcal{T}$ on $G_{k,\mathcal{T}}$ and $G'_{k,\mathcal{T}}$.

The graphs $G_{k,\mathcal{T}}$ and $G'_{k,\mathcal{T}}$ are constructed as follows. There are n constants $\{1, ..., n\}$ in the domain, where n is even. The two graphs have the same set of $(2^{n/2} + 1)$ nodes. Each node is a set of constants. One of the nodes α is a set containing all the constants in the domain n. All the other $2^{n/2}$ nodes are subsets of cardinality $\frac{n}{2}$ of the domain. The $2^{n/2}$ nodes are distributed in two classes of equal cardinality, In_n and Out_n, in a regular way described below. The graphs have a star shape (see Figure 5), where $\alpha = \{1, ..., n\}$ is the central node, and is linked to all other nodes. There are no other nodes, and no other edges than the ones between α and any other node. In $G_{k,\mathcal{T}}$, the node α has the same in- and out-degree. More precisely, there is a vertex from each node in In_n to the central node, α, and a vertex from the central node, α, to each node in Out_n. In $G'_{k,\mathcal{T}}$, one of the edges is inverted so that the in-degree of α is bigger than its out-degree.

We next explain how the sets of nodes In_n and Out_n are constructed. Let $\mathcal{P}_{n/2}(n)$ be the set of subsets of cardinality $\frac{n}{2}$ of the domain $\{1 \ldots n\}$. We choose $In_n \subset \mathcal{P}_{n/2}(n)$ and $Out_n \subset \mathcal{P}_{n/2}(n)$ such that they satisfy the following probabilistic property for each $i \in n$:

$$p(i \in S | S \in In_n) \;=\; p(i \in S | S \in Out_n) \;=\; \frac{1}{2}. \tag{1}$$

The existence of such In_n and Out_n is proved as follows:
Basis: For $n = 4$, we consider $In_n = \{\ \{1,2\}, \{3,4\}\ \}$ and $Out_n = \{\ \{1,3\}, \{2,4\}\ \}$.
Induction: Suppose that In_n and Out_n satisfy the property. Then, In_{n+2} and Out_{n+2} as defined as follows satisfy the requirements.

$$In_{n+2} = \{S \cup \{n+1\} | S \in In_n\} \cup \{S \cup \{n+2\} | S \in Out_n\},$$

$$Out_{n+2} = \{S \cup \{n+1\} | S \in Out_n\} \cup \{S \cup \{n+2\} | S \in In_n\}.$$

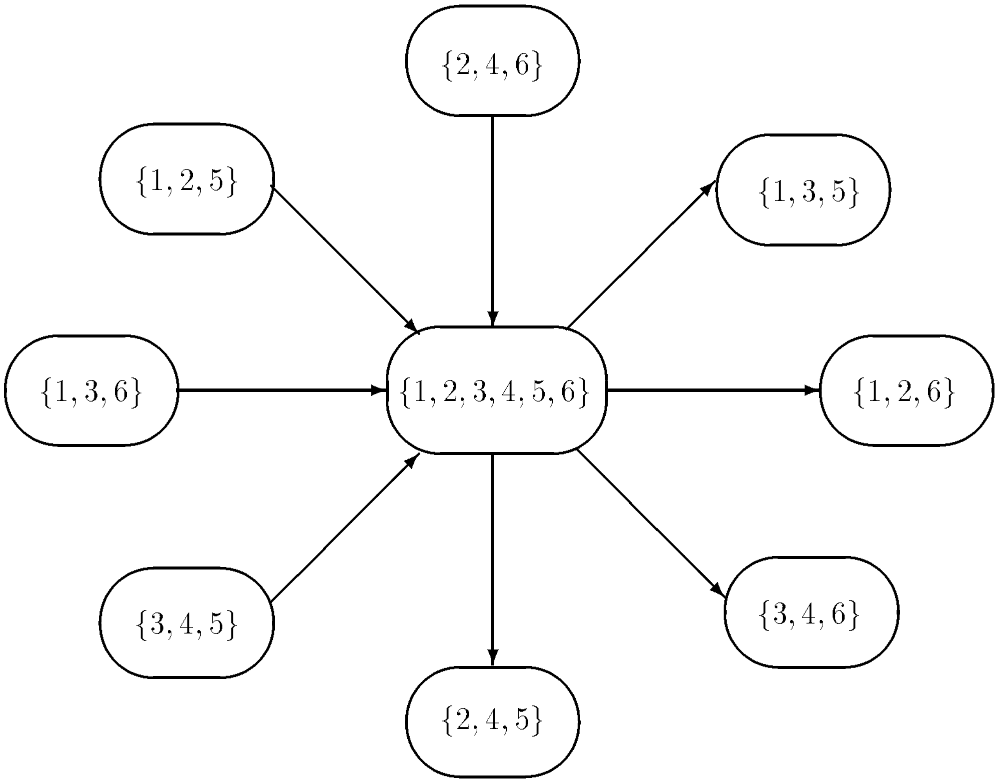

Figure 1: The graph $G_{k,\mathcal{T}}$ for $n = 6$

We next show that the duplicator has a winning strategy on the two graphs if n is big enough. Assume first that $\mathcal{T} = \{U, \{U\}\}$, so there are no types other than the atomic type and the type of the nodes. At each step, the spoiler can choose either an atomic constant or a set of atomic constants (since $\mathcal{T} = \{U, \{U\}\}$). Consider the set of permutations on n, which can be extended to isomorphisms on the substructures defined by the chosen objects. We show that there is always such a permutation left at each step of the game. After each move, the number of possible isomorphisms between the substructures defined by the chosen objects decreases. As long as there is an isomorphism left, the duplicator wins the move. The most efficient strategy for the spoiler to decrease the number of possible isomorphisms, is to always choose sets of size $\frac{n}{2}$.

At the beginning, there are $n!$ possible isomorphisms. If the spoiler chooses one element, or a singleton, or a set of $n-1$ elements, the number of isomorphisms left after the duplicator's move, is $(n-1)!$. The most important decrease of the number of isomorphisms left is obtained if the spoiler chooses a set of cardinality $\frac{n}{2}$, in which case, there are $\frac{n}{2}!$ isomorphisms left. By induction on the number of moves, the number of possible matchings left after the k^{th} move is at least $(\frac{n}{2^k}!)^{2^k}$.

It follows from Property (1) that the duplicator will be able to find a matching set in both In_n and Out_n, and make the appropriate choice. Therefore, the duplicator has a winning strategy if $n > 2^k$.

For the general case where $\mathcal{T}$ contains other types, for instance tuples of arity l, then the number of isomorphisms decreases faster. In the worst case, by choosing a set of tuples of arity l, the spoiler could decrease the set of isomorphisms as if he had chosen successively l times, sets of atomic constants of arity $\frac{n}{2}$. Therefore, the duplicator has a winning strategy if $n > 2^{k \times l}$, where l is the largest arity of the types in $\mathcal{T}$. $\Box$

The powerbag operator

The complexity of queries in BALG is highly related to the definition of the powerset operator. We next consider the alternative operator for the powerset, called the *powerbag*, and denoted by $\mathcal{P}_b$. The powerbag is similar to the powerset, except that it distinguishes between different occurrences of the same element. Its output is a bag with duplicates.

Definition 5.1 Let B be a bag of type $\{\!\{T\}\!\}$. $\mathcal{P}_b(B)$ is a bag of type $\{\!\{\{\!\{T\}\!\}\}\!\}$, defined as follows: let h be a mapping that maps each occurrence of each constant in B to a different (new) constant, and let H be its natural extension to bags of constants, then $\mathcal{P}_b(B) = H^{-1}(\mathcal{P}(H(B)))$.

For example, the powerbag of $\{\!\{a, a\}\!\}$ differs from its powerset.

$$\mathcal{P}_b(\{\!\{a, a\}\!\}) = \{\!\{\{\!\{\}\!\}, \{\!\{a\}\!\}, \{\!\{a\}\!\}, \{\!\{a, a\}\!\}\}\!\},$$

while

$$\mathcal{P}(\{\!\{a, a\}\!\}) = \{\!\{\{\!\{\}\!\}, \{\!\{a\}\!\}, \{\!\{a, a\}\!\}\}\!\}.$$

The powerbag is the most natural operation in presence of bags. Nevertheless, we show below that it results in a dramatic increase of the complexity of the algebra. This justifies the choice of the powerset instead.

Recall from Proposition 3.2, that the explosion of the number of duplicates created by the successive applications of $\mathcal{P}$ and δ, is exponential for the first step and becomes polynomial afterwards. This is the fundamental tool in the proof of Theorem 5.1. In contrast, iterative applications of $\delta(\mathcal{P}_b(B))$ create an exponential number of duplicates at each step. (This is because $\mathcal{P}_b$ distinguishes between different occurrences of the same element.) This difference has a strong effect on the complexity of the language.

Theorem 5.5 For every i, there is a query in BALG^2+$\mathcal{P}_b$, with $hyper(i)$-TIME complexity.

The theorem is proved by showing that every $hyper(i)$ function over integers can be encoded in BALG^2. The proof is based on an encoding of arithmetic functions in terms of bags. We first establish a technical correspondence between BALG^2 and arithmetic, by showing the close relationship between queries in BALG^2, and formulas in number theory $(N, +, \times, \leq, 0, 1)$.

We start by defining arithmetic formulas with bounded quantifications.

Definition 5.2 Let $\phi(x) = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n F(x_1, \ldots, x_n, x)$ be a formula in prenex normal form in the arithmetic, with the quantifiers Q_i, $i = 1..n$, the matrix F, and where x is the only free variable in ϕ. We say that $\phi(x)$ is **restricted by the function** f iff for every n, $\phi(n)$ is true, iff $\phi_f(n)$ is true, where

$$\phi_f(x) = Q_1 x_1 < f(x)\ Q_2 x_2 < f(x)\ \ldots Q_n x_n < f(x)\ F(x_1, \ldots, x_n, x).$$

The next result shows that the computation of a bounded complexity Turing machine can be encoded in a formula in the arithmetic with bounded quantification.

Lemma 5.6 Let M be some $f(x)$-time bounded Turing machine. There exist a polynomial P and an arithmetic formula $\phi(x)$ restricted by $2^{P(f(x))}$, such that for every integer w,

$$M \text{ accepts } w \text{ iff } \phi(w) \text{ is true.}$$

Proof : The proof follows from the fact that a computation of length $f(n)$ can be encoded by an integer i in the range $0 \leq i < 2^{P(f(n))}$. It is shown in [HU79] that a computation of length $f(n)$ can be represented by $((f(n)+1)^2+1)$ k-ary digits where k is the number of symbols of the machine. It was also shown that this sequence of digits can then be encoded by an integer i in the range $0 \leq i < 2^{P(f(n))}$, where the first digits are the input word w. Furthermore, it is proved there that the statement "M accepts w" can be expressed by an arithmetic formula of the form $\phi(w) = \exists i(E(i,w))$, where E is a predicate that is true iff the integer i is an encoding of a successful computation of M over w. Since M is an $f(n)$-time bounded Turing machine, $\phi(w)$ is true iff there exists an $i < 2^{P(f(n))}$ that is an encoding of a successful computation. Thus $\phi(w)$ is true iff $\phi_{2^{P(f(n))}}(w) = \exists i < 2^{P(f(n))}(E(i,w))$ is true. □

In the next result, we show that properties described by arithmetic formulas that are restricted by some hyper-exponential functions can be expressed in BALG^2.

Lemma 5.7 For every arithmetic formula $\phi(x)$ restricted by some hyper-exponential function $hyper(i)$, there exists an expression ϕ' in BALG$^2 + \mathcal{P}_b$ such that for every n,

$$\phi(n) \text{ is true iff } \phi'(b_n) \text{ is not empty,}$$

where b_n is a bag of size n containing n occurrences of a single constant a.

Proof : The idea is to simulate integers using bags. We encode an integer i by a bag containing i copies of the element a. Addition of integers is simulated by $\uplus$, and multiplication by $\times$. Bounded quantification over integers is simulated using nested bags. The bounded domain is represented by a nested bag D containing bags of size 1 to $hyper(i)(n)$ where n is the input integer. We first explain below how this nested bag is constructed. Given that we explain how arithmetic formulas are translated to the algebra.

We start by showing how to construct the bag representing the bounded domain of integers used in ϕ. Given a bag b_n containing n occurrences of a, the following formula constructs a bag containing 2^n occurrences of a, $E(b_n) = \pi_2(\mathcal{P}_b(b_n) \times \{\!\{[a]\}\!\})$. The bag containing all bags representing the integers $1 \ldots hyper(i)(n)$ is constructed as follows, $D(b_n) = \mathcal{P}(E^i(b_n))$.

We next show how arithmetic formulas are translated to the algebra. We follow the lines of the classical translation from calculus to algebra [Ull88, AB87, BM92], and use induction on the structure of the arithmetic formula. We present only points which are specific to our case.

We assume w.l.o.g. that ϕ does not contain the symbol $\leq$ (since $\leq$ is easily expressed by $+$ and $=$). ϕ' is defined inductively as follows.

Basis: An atomic formula of the form $e_1 = e_2$, where e_1, e_2 are arithmetic expressions with free variables $x_1, \ldots, x_m$, is translated to the BALG^2 expression $\sigma_{\hat{e_1}=\hat{e_2}}(D_1 \times \ldots \times D_m)$, where $D_i = D(b_n)$ if x_i is not the input variable n, and $D_i = \{\!\{b_n\}\!\}$ otherwise. $\hat{e}_i$ $(i = 1, 2)$ is constructed from e_i by (i) replacing every occurrence of variable x_j in e_i by the projection of the j^{th} attribute of tuples in $D_1 \times \ldots \times D_m$, and (ii) replacing every occurrence of $+$ and $\times$ (for integers) by by $\uplus$ and $\times$ (for bags) respectively.

For example, consider the equation $x_1 + x_2 = x_1 \times x_3 + n$. It is translated to BALG^2 as follows:

$$\sigma_{\lambda x, \alpha_1(x) \uplus \alpha_2(x) = \alpha_1(x) \times \alpha_3(x) \uplus \alpha_4(x)}(D(b_n) \times D(b_n) \times D(b_n) \times \{\!\{b_n\}\!\}).$$

Induction: For the induction step it suffices to consider cases where ϕ is constructed from atomic formulas using $\wedge, \neg$ and existential quantifier. For $\wedge$ we use Cartesian product and selection by equality for variables with the same names, and then use projection (using MAP and duplicate elimination) for omitting multiple occurrences of the same variables.
For negation we take the complement with respect to the Cartesian product of the sets $D(b_n)$ that represent the domains of the free variables in the negated formula. Finally we use projection (using MAP and duplicate elimination) for the existential quantifier. $\Box$

It follows that if M is some hyper-exponential time bounded Turing machine. Then there exists a BALG$^2 + \mathcal{P}_b$ formula such that for every w,

$$M \text{ accepts } w \text{ iff } \phi(b_w) \text{ is not empty},$$

where b_w is a bag of size corresponding to the integer representation of w. This concludes the proof of Theorem 5.5. Other properties of queries using the powerbag operator are further discussed in the next section.

6 More Nesting, BALG^3, and BALG^k

It turns out that adding one more level of bag nesting increases dramatically the expressive power of the language. We start by considering queries with unnested input/output type. Next we investigate queries over nested input/output.

Recall that $\mathcal{E}$ denotes the set of *elementary queries*.

Theorem 6.1 BALG$^3 = \mathcal{E}$ over unnested bags.

Note that with three levels of bag nesting, BALG^3 differs radically from RALG^3, which is only in EXPSPACE. This is due to the fact that in BALG^3, two successive application of powerset, $\mathcal{P}$, followed by two successive application of bag-destroy, δ, lead to an exponential increase of the number of duplicates. This follows from Proposition 3.2. Due to the type limitation, it was not possible in BALG^2, to apply two times consecutively the powerset operator. We next prove Theorem 6.1.

Proof : Clearly, every BALG^3 query is in some hyper-exponential complexity. Indeed, every operation of BALG^3 increases at most exponentially the size of its input, and there are finitely many operations in an expression.

We next show that every $hyper(i)$-time bounded query over unnested bags can be expressed in BALG^3. The proof is based on an encoding of the computation of a Turing machine M in a bag. The technique is very similar to the one used in [HS91]. A computation is represented by a bag containing 4-attribute tuples of type $[\{\!\{U\}\!\}, \{\!\{U\}\!\}, U, U]$. The first two attributes are used as indices for the time and the space respectively (an integer i is represented by a bag b_i containing i occurrences of a given constant). A tuple $[b_i, b_j, a, q]$ is in the bag iff in the i^{th} step of the computation of M, the j^{th} cell of the tape contains the symbol a, and the head of the machine is on that cell in state q (if the head is elsewhere then instead of q we have a special constant $\square$). To encode a computation of $hyper(i)$-time complexity over an input of size n, bags of sizes 1 to $hyper(i)(n)$ are needed for the time and space indices. These bags are constructed as follows. Given a bag B of size n, the expression $N(B) = \pi_1(\{\!\{[a]\}\!\} \times B)$ constructs a bag containing n occurrences of a. The next expression defines a bag containing 2^n occurrences of a, $E(B) = N(\mathcal{P}(\mathcal{P}(N(B))))$. Finally, a bag containing all bags of sizes 1 to $hyper(i)(n)$ is constructed by $D(B) = \mathcal{P}(E^i(B))$.

The algebra expression simulating the computation of the Turing machine is similar to the one used in [HS91, AHV94]. We first construct a big set containing all 4-ary tuples that may represent part of the computation (to do so, we take the Cartesian product of the domain of indexes, domain of tuples and domain of states). Then we take the powerset, (thus we construct all possible subsets of 4-ary tuples). Finally, we select the bags representing a legal terminating computation. We explain below how this algebra expression is constructed. We present only elements which are special to our case.

Let $A = \{\!\{0, 1, [,], \#, \{, \}\}\!\}$ be a bag containing the alphabet of the machine. Let Q be a bag containing the states of the Turing machine, and an additional new symbol $\square$. Let $q_0, q_f \in Q$ be respectively the initial and final states of the machine. Let $M(B)$ be a bag representing all the possible changes in the tape of the Turing machine, caused by legal moves of the machine. $M(B)$ is a bag of type $\{\!\{[\{\!\{[U, U, U]\}\!\}, \{\!\{[U, U, U]\}\!\}]\}\!\}$, i.e. a bag of 2-ary tuples where both attributes are of type $\{\!\{[U, U, U]\}\!\}$. The first attribute describes a set of cells before some change, and the second attribute describe the same cells after the change. The set of modified cells is represented using a bag of 3-ary tuples. Each tuple corresponds to one cell. The first attribute is the location on the tape, the second is the content of the cell, and the third indicates if the head of the tape is in that location or not, and in which state. $M(B)$ is defined as follows:

$$M(B) = M_{\lambda_1}(B) \uplus \ldots \uplus M_{\lambda_m}(B),$$

where $\lambda_1, \ldots, \lambda_m$ are the legal moves of the machine. If λ_i is a move of the form $\lambda(a_1, q_1) = (R, a_2, q_2)$ then

$$M_{\lambda_i}(B) = MAP_{\lambda y, [\{\!\{[y, a_1, q_1], [y \uplus \{\!\{[a]\}\!\}, b, \square]\}\!\}, \{\!\{[y, a_2, \square], [y \uplus \{\!\{[a]\}\!\}, b, q_2],\}\!\}]} D(B).$$

In other words, for each possible space stamp y in $D(B)$, $M_{\lambda_i}(B)$ contains a tuple representing the potential move λ_i, encoded as a binary tuple [before the move , after the move] of partial configurations in cell y and the following cell $y \uplus \{\!\{[a]\}\!\}$, of the form:

$$[\{\!\{[y, a_1, q_1], [y \uplus \{\!\{[a]\}\!\}, b, \square]\}\!\}, \{\!\{[y, a_2, \square], [y \uplus \{\!\{[a]\}\!\}, b, q_2],\}\!\}].$$

Else, if λ_i is a move of the form $\lambda(a_1, q_1) = (L, a_2, q_2)$ then

$$M_{\lambda_i}(B) = MAP_{\lambda y,[\{\!\{[y \uplus \{\!\{[a]\}\!\},a_1,q_1],[y,b,\Box]\}\!\},\{\!\{[y \uplus \{\!\{[a]\}\!\},a_2,\Box],[y,b,q_2],\}\!\}]} D(B).$$

Let $enc(B)$ be a bag of bags, where each $b \in enc(B)$ is a possible encoding of the initial state of the machine, for the input bag B. Every $b \in enc(B)$ is of type $\{\!\{[\{\!\{U\}\!\}, U, U]\}\!\}$. The first attribute denotes the location of a cell on the tape, the second attribute is the content of the cell, and the third contains q_0 for the first cell and $\Box$ for the other cells. The construction of $enc(B)$ is standard (for detailed description see [HS91, AHV94]). We first "guess" an order on the constants occurring in B (using the powerset operator), then we use this order to give binary representation to each constant and to define an order on the tuples. Finally, we list on the tape the encoding of each tuple according to the defined order. The only difference from the classic construction presented in [HS91, AHV94], is that instead of encoding the location on the tape using ordered tuples/sets, we use here bags of constants, i.e. the i^{th} location is denoted using a bag of size i.

The formula simulating computations of $hyper(i)$ time, on an input B has the form:

$$TM(B) = \sigma_{\phi_1}\sigma_{\phi_2}\sigma_{\phi_3}(\mathcal{P}(D(B) \times D(B) \times A \times Q)),$$

where

$$\begin{array}{rcl}
\phi_1(x) & = & (\pi_{2,3,4}\sigma_{\lambda y, y.1=\{\!\{\}\!\}}(x)) \in enc(B), \\
\phi_2(x) & = & (\mathrm{MAP}_{\lambda y,[(y.1-y.2),(y.2-y.1)]} \\
& & \quad (\mathrm{MAP}_{\lambda y,[\pi_{2,3,4}(\sigma_{\lambda z, z.1=y.1}(y.3)),\pi_{2,3,4}(\sigma_{\lambda z,z.1=y.2}(y.4))]} \\
& & \qquad (\sigma_{\lambda y, y.1 \uplus \{\!\{[a]\}\!\}=y.2}(D(B) \times D(B) \times x \times x)))) \subseteq M(B), \\
\phi_3(x) & = & (\sigma_{\lambda y, y.4=q_f}(x)) \neq \{\!\{\}\!\}.
\end{array}$$

The first selection ϕ_1 checks if at the beginning of the computation (time 0) the tape contains an encoding of the input bag B, and the head is on the first cell of the tape in an initial state. The second selection ϕ_2 checks if the changes in the content of the tape in two consecutive steps corresponds to a legal move of the machine. Finally, the last selection ϕ_3 checks if the computation reached an accepting state. $\Box$

The space needed for computing a query Q depends on the type of operations used in Q. In particular, it turns out that the space complexity is highly related to the number of powerset operations used in Q.

Every BALG query can be viewed as a tree with nodes representing operations, and leaves representing bags and constants. The **power nesting** of an expression is the maximal number of powerset operations in a path from the root to a leaf. Let BALG_i^k be the class of queries expressible by BALG^k expressions with power nesting less or equal to i. An input such that the number of distinct elements is proportional to the size is called a *sparse input*.

Theorem 6.2 .

- $hyper(\lfloor \frac{i}{2} \rfloor - 1)$-NTIME $\subseteq \mathrm{BALG}_i^3 \subseteq hyper(\lfloor \frac{i}{2} \rfloor)$-SPACE.
- $hyper(\lfloor \frac{i}{2} \rfloor)$-NTIME $\subseteq \mathrm{BALG}_i^3 \subseteq hyper(\lfloor \frac{i}{2} \rfloor)$-SPACE, for sparse inputs.

Proof : The inclusion of $hyper(\lfloor\frac{i}{2}\rfloor-1)$-NTIME in BALG^3_i follows from the proof of Theorem 6.1. The expression used there to encode $hyper(i)$ time bounded TM computation contains exactly $2i+2$ nested powerset applications. The operator was applied $2i+1$ times for constructing the domain of indices, and then one more time for constructing all the possible subsets of 4-ary tuples.

Note that a non sparse input of size n may contain n occurrences of the same tuple, thus two consecutive applications of powerset may increase the size of the output by only one exponential. For sparse input, the first two powersets cause double exponential growth. Thus to construct a bag containing all bags of size 1 to $hyper(i)(n)$ we can use the expression $\mathcal{P}(E^{i-2}(\mathcal{P}(\mathcal{P}(B))))$. (where E is defined as in the proof of Theorem 6.1). This expression contains only $2i-1$ nested powerset operations. To encode the Turing machine we use one additional powerset. Thus $2i$ powersets are sufficient for encoding a $hyper(i)$ time computation for sparse input.

We next prove the inclusion of BALG^3_i in $hyper(\lfloor\frac{i}{2}\rfloor)$-SPACE. A simple combinatorial argument shows that except for the first powerset operation, the only expressions that increase the size of their input exponentially are expressions consisting of two nested applications of powerset with no occurrence of bag destroy between them. The reason is that (i) all the operations except powerset gives only polynomial growth, and (ii) as shown in the proof of Proposition 3.2, except for the first powerset, a single application of powerset followed by bag destroy gives only polynomial growth. Thus, to study the space complexity of BALG^3_i it is sufficient to consider expressions of the form $(\delta\delta\mathcal{P}\mathcal{P})^k(B)$, $\delta\mathcal{P}\mathcal{P}(\delta\delta\mathcal{P}\mathcal{P})^{k-1}(B)$, $\mathcal{P}\mathcal{P}(\delta\delta\mathcal{P}\mathcal{P})^{k-1}(B)$, and $\mathcal{P}(\delta\delta\mathcal{P}\mathcal{P})^{k-1}(B)$, for $k\leq\lfloor\frac{i}{2}\rfloor$.

From Proposition 3.2 it follows that the number of occurrences of each tuple in the intermediate results of such expressions is at most $hyper(\lfloor\frac{i}{2}\rfloor+1)(n)$. Since the number of different tuples is polynomial in the size of the input, every unnested bag used in the computation can be encoded in $hyper(\lfloor\frac{i}{2}\rfloor)(n)$-SPACE. From the proof of the proposition it follows that the number of different unnested bags in the intermediate results it at most $hyper(\lfloor\frac{i}{2}\rfloor)(n)$, and that the number of occurrences of each such bag is at most $hyper(\lfloor\frac{i}{2}\rfloor+1)(n)$. Thus every nested bag (bag of bags of tuples) used in the computation can be encoded in $hyper(\lfloor\frac{i}{2}\rfloor)(n)$ space.

The rest of the proof is classical and is done as in Theorem 5.1. □

Increasing the nesting of powerset thus strictly increases the expressive power of the queries. Theorem 6.2 implies that $\text{BALG}^3_i\subset\text{BALG}^3_{(i+2)}$. We could not come up with a better hierarchy.

The previous result can be generalized as follows in the case of BALG^k.

Proposition 6.3

- $hyper(i-2)$-NTIME $\subseteq\text{BALG}^k_{\frac{k-1}{k-2}i}\subseteq hyper(i)$-SPACE.
- $hyper(i-1)$-NTIME $\subseteq\text{BALG}^k_{\frac{k-1}{k-2}i}\subseteq hyper(i)$-SPACE, for sparse inputs.

Proof : The proof is very similar to that of Theorem 6.2. The only difference is that now we can apply $k-1$ consecutive powersets. We explain how this affects the proof.

The inclusion of $hyper(i-2)$-NTIME in $\mathrm{BALG}^k_{\frac{k-1}{k-2}i}$ follows from the fact that the indices used to simulate a computation of $hyper((k-2)i)$-time complexity can be constructed by an expression of the form $D(B) = \mathcal{P}(E^i(B))$, where $E(B) = N(\mathcal{P}^{k-1}(N(B)))$, and $N(B) = \pi_1(\{\!\{[a]\}\!\} \times B)$. To simulate the Turing machine we need one additional powerset application, Thus, we need $(k-1)i+2$ nested powersets to simulate a $hyper((k-2)i)$-time complexity computation. For sparse input, the same $D(B)$ computes enough indices for a computation of $hyper((k-2)i+1)$-time complexity.

The inclusion of $\mathrm{BALG}^k_{\frac{k-1}{k-2}i}$ in $hyper(i)$-SPACE follows from the fact that the number of occurrences of each constant in $(\delta^{k-1}\mathcal{P}^{k-1})^i(B)$ is at most $hyper((k-2)i+1)(|B|)$. (The proof is the same as that of Proposition 3.2). Thus every bag used in the computation of $(\delta^{k-1}\mathcal{P}^{k-1})^i(B)$ can be encoded in $hyper((k-2)i)$-SPACE. □

Note that if the powerbag operation $\mathcal{P}_b$ (presented in Section 5) is added to the language, then every application of $\mathcal{P}_b$ may increase the size of the input exponentially. $\mathrm{BALG}^k_i + \mathcal{P}_b$ is the natural extension of BALG^k_i with powerbag such that the nesting of powerbag or powerset operators is at most i. Thus, we have that:

Proposition 6.4 .

- $hyper(i-2)$-NTIME $\subseteq \mathrm{BALG}^k_i + \mathcal{P}_b \subseteq hyper(i-1)$-SPACE.

Proof : Here, again, we use the same proof technique as above. The only difference is that now each application of powerbag increases the size of the output exponentially.

The inclusion of $hyper(i-2)$-NTIME in $\mathrm{BALG}^k_i + \mathcal{P}_b$ follows from the fact that the indices used to simulate a computation of $hyper(i)$-TIME complexity can be constructed by an expression of the form $D(B) = \mathcal{P}(E^i(B))$, where $E(B) = \pi_1(\{\!\{[a]\}\!\} \times \mathcal{P}_b(B))$. To simulate the Turing machine we need one additional powerset application, Thus, we need $i+2$ nested powersets and powerbags to simulate a $hyper(i)$-time complexity computation.

The inclusion of $\mathrm{BALG}^k_i + \mathcal{P}_b$ in $hyper(i-1)$-SPACE follows from the fact that the number of occurrences of each constant in the output of a $\mathrm{BALG}^k_i + \mathcal{P}_b$ expression is at most $hyper(i)$ in the size of the input. Thus every bag used in the computation can be encoded in $hyper(i-1)$-SPACE. □

The last subject we consider is queries over *nested inputs*. We do not restrict our attention only to BALG^3, but instead present a more general result.

Theorem 6.5 For every $k \geq 3$, BALG^k expresses exactly all the elementary queries over inputs and outputs of bag nesting $(k-1)$.

Proof : Every BALG^k query is in some hyperexponential complexity, since every operation increases at most exponentially the size of its input.

The proof that BALG^k expresses all the elementary queries over inputs and outputs of bag nesting $(k-1)$ is based on an encoding of the computation of a Turing machine M in a bag. The algebra expression used to simulate the computation is identical to that of

Theorem 6.1, except that the encoded input/output is now nested. The additional bag nesting is needed to enable encoding. The encoding/decoding is the classical one. It is fully described in [HS91], and is therefore omitted here. □

Fixpoint operators have been extensively studied [GS86] in the context of query languages, in relation with recursion. The inflationary fixpoint operator, IFP, of an algebraic expression ϕ is defined as the least fixpoint of the operator: $T(B) = \phi(B) \cup B$. The next result shows that with fixpoint, the algebra is Turing complete.

Theorem 6.6 For every $k \geq 2$, BALG^k+IFP is Turing complete.

Proof : We have to show that BALG^k+IFP can simulate every computable query. Let B be some input bag. By definition, there exists a Turing machine M which, on input $enc(B)$ terminates with output $enc(q(B))$, where enc is some function that maps bags to an encoding on the tape of the Turing machine. Thus, we will construct a BALG^k+IFP formula ϕ which, on input B does the following:

- encode B into $enc(B)$,
- simulate the computation of M on input $enc(B)$, and
- decode the result $q(B)$ from its encoding $enc(q(B))$.

The encoding/decoding is done as in Theorem 6.1 (see also [HS91, AHV94]). We next focus on the simulation of M. We need to represent a configuration of M as a bag. In particular, the tape has to be represented. Since the tape is infinite, we only represent at each step a finite portion, which is the portion used so far. As in the proof of Theorem 6.1 the computations are represented in a bag containing 4-ary tuples where the first two attributes are used as indices for time and location. In a bag describing the computation up to time t, the highest time index is a bag of size t. We start with a bag representing the configuration of the machine at the initial state (i.e. at time 0). Then at each iteration we compute the new configuration of the machine and add the tuples representing it to the bag. Note that one cannot remove the tuples representing old configurations of M, due to the inflationary nature of BALG^k+IFP computations. Thus, the first attribute is used as time stamp that keeps track of the sequence of configurations in a computation of M. Note that the fact that we represent the indices of time and space using bags enables us to use indices of unbounded size.

We are dealing now with a "double" encoding: the database is encoded on the tape $enc(B)$, then the tape is represented (encoded) by a bag $enc(B)$. For a given instance B, the simulation of M proceeds in two phases:

(†) compute a representation $enc(B)$ of the initial configuration of M on input B.

(‡) compute the sequence of consecutive configurations of M until termination.

The construction of $enc(B)$ in (†) is essentially the same as that in the proof of Theorem 6.1. We next outline the construction for (‡). One has to simulate the computation of M starting from the initial configuration represented in $enc(B)$. To construct a new configuration from the current one, one has to simulate a move of M. This is repeated until M reaches a final state (accepting or rejecting). The iteration can be performed using the fixpoint operator in BALG^k+IFP. Each step consists of defining the new configuration from the current one, timestamping it, and adding it to the current bag. This can be done with a BALG^k+IFP formula.

For instance, suppose the current state of M is q, the content of the current cell is 0, and the corresponding move of M is to change 0 to 1, move right, and change states from q to r. Let B_M be a bag describing the computation up to time t, and assume it contains a tuple $[t, j, 0, q]$. The tuples describing the new configuration of M are the tuples $[t \uplus \{\{[a]\}\}, i, x, y]$ such that:

(a) $i \subset j$ or $i \supset j \uplus \{\{[a]\}\}$, and $[t, i, x, y] \in B_M$, or

(b) $i = j$, $x = 1$, and $y = \Box$, or

(c) $i = j \uplus \{\{[a]\}\}$ and $y = r$ and $[t, i, x, \Box] \in B_M$.

In other words, (a) says that the cells other than the j-th cell and its successor remain unchanged; (b) says that the content of cell j changes from 0 to 1, and the head no longer points to the j-th cell; finally, (c) says that the head points to the successor of the jth cell, the new state is r, and the content x of the cell is unchanged. Clearly, (a)-(c) can be expressed by a BALG^k+IFP formula. One such formula is needed for each instruction of M, and the formula corresponding to the finite set of instructions is obtained by their union.

With (†) and (‡) achieved, it remains to decode the representation of $enc(q(B))$ in B_M to obtain the result. This is essentially the inverse of the encoding process. It can be easily verified that this can be achieved using a BALG^k+IFP formula. $\Box$

Theorem 6.6 constitutes a negative result. Nevertheless, fixpoint can be added to the bag algebra with less dramatic consequences for the complexity, by using bounded fixpoint [Suc93]. Bounded fixpoints were introduced in the context of nested sets. They can be easily adapted to nested bags, and lead to an increase of expressive power, while keeping a bounded complexity. Transitive closure is expressible in the extension of BALG^1 to bounded fixpoint.

7 Conclusion

Many database systems use bags to implement relations. Moreover, in practical query languages (e.g., SQL), some operations (e.g., aggregate functions such as COUNT, AVG) are sensitive to the number of duplicates. We studied an algebra for bags, which extends the relational algebra. Interestingly, we proved that without bag nesting it constitutes a tractable query language (LOGSPACE).

Power of data types From a theoretical point of view, the results show the impact of the types manipulated by a language on the expressive power and the complexity. We proved that for both unnested bags ($BALG^1$) and nested bags with only one level of nesting ($BALG^2$), the complexity is very similar to that of the relational algebra and the nested set algebra respectively. On the other hand, the expressive power is increased, and practical queries (such as cardinality comparison) which were not definable with set semantics, become definable with the bag semantics. The bags essentially give the ability to count. Usage of other data types give rise to new definable queries as shown in [GM95].

Nest vs. powerset In this paper, we considered very powerful primitives to deal with nested bags, such as the powerset. Weaker primitives were proposed in the case of nested sets, such as the set-nesting operator, *nest*. It was shown in [PG88, PG92], that, in the nested relation algebra with no powerset but a nest operation, the set nesting of intermediate types doesn't increase the expressive power of the algebra for relational queries. Conservative extension properties were shown to carry over [Won93] in a more general setting with sets, bags and lists for queries over nested inputs and outputs. In particular, it was shown that the use of intermediate types higher than the bag nesting of both the input and the output, doesn't increase the expressive power of the bag algebra without the powerset but with the nest operator, $\mathrm{BALG} \cup \{\mathrm{nest}\} - \{\mathcal{P}\}$. It follows that the results comparing the expressive power of the nested relational algebra with the nested bag algebra, carry over for this new paradigm. In particular, we have: $\mathrm{RALG}^2 \cup \{\mathrm{nest}\} - \{\mathcal{P}\} \subset \mathrm{BALG}^2 \cup \{\mathrm{nest}\} - \{\mathcal{P}\}$.

Optimization It has been shown in [CV93] in particular that optimization techniques for conjunctive queries under a set semantics do not carry over under a bag semantics. It is unclear if having bags as first class citizens (instead of just an implementation tool over which the user has no control) allows to write more efficient queries, or to get better optimization tools. Several classical aggregate functions are expressible directly in the algebra. The user can trigger duplicate elimination. Does it optimize?

Objects and oid's Moreover, nested bags can be used to simulate objects and oid's, and therefore they can be used to study properties of object oriented languages. We don't consider this aspect in the paper. Bags are very similar to sets containing of objects. Bags contain several occurrences of the same element. Similarly, sets of objects may contain several objects with the same state. The main difference is in the fact that objects have oid's, while bags contain pure values. Is this significant ? It turns out that the answer is positive. In particular, it affects the expressive power of languages. For example, the abstraction operation (for objects) and the duplicate elimination operations (for bags) have similar effects. They both eliminate elements having the same value/state. But while a restricted version of abstraction can be expressed in object oriented languages [BP91], we showed that duplicate elimination cannot be expressed by the value oriented bag language.

References

[AB87] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. In *Proc. Int. Workshop on Theory and Applications of Nested Relations and Complex Objects (extended abstract)*, Darmstadt, 1987. INRIA research report n 846.

[AFS89] S. Abiteboul, P.C. Fischer, and H.-J. Schek. *Nested Relations and Complex Objects*. LNCS 361. Springer-Verlag, 1989.

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1994.

[Alb91] J. Albert. Algebraic properties of bag data types. In *Proc. 17th Int'l Conf. on Very Large Data Bases*, pages 211–219, 1991.

[AV91] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proc. ACM Symp. on Theory of Computing*, New Orleans, May 1991.

[BBN91] V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In *Proc. 3rd Int. Workshop on database programming languages*, pages 9–19, Nafplion, Aug. 1991. Morgan Kaufman.

[BK90] C. Beeri and Y. Kornatzky. Algebraic optimization of object oriented query languages. In *Proc. 3rd Intl. Conf. on Database Theory, ICDT 90, Paris, France*, 1990.

[BM92] C. Beeri and T. Milo. Functional and predicative programming in oodb's. In *Proc. 11th Symp. on Principles of Database Systems San-Diego*, 1992.

[BM93] C. Beeri and T. Milo. On the power of algebras with recursion. In *Proc. ACM SIGMOD Conference on Management of Data*, 1993.

[BP91] J. Van Den Bussche and J. Paredaens. The expressive power of structured values in pure OODB's. In *Proc. 10th Symp. on Principles of Database Systems*, 1991.

[BS91] V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with sets/bags/lists. In *Proc. 18th Int. Col. on Automata, Languages and Programming*, 1991.

[BTBW92] V. Breazu-Tannen, P. Buneman, and L. Wong. Naturally embedded query languages. In *Proc. of Intl. Conf. on Database Theory*, pages 140–154. LNCS, Springer Verlag, 1992.

[CDV88] M. Carey, D. DeWitt, and S. Vandenberg. A data model an query language for exodus. In *Proc. ACM SIGMOD Conference on Management of Data*, 1988.

[CH80] A. Chandra and D. Harel. Computable Queries for Relational Data Bases. *Journal of Computer and System Sciences*, 21(2):156–178, Oct. 1980.

[CV93] S. Chaudhuri and M. Vardi. Optimization of real conjunctive queries. In *Proc. 12th ACM Symp. on Principles of Database Systems*, Washington, May 1993.

[DGK82] U. Dayal, N. Goodman, and R.H. Katz. An extended relational algebra with control over duplicate elimination. In *Proc. ACM Symp. on Principles of Database Systems, Los Angeles, CA,*, 1982.

[Ehr61] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49, 1961.

[Fag75] R. Fagin. Monadic generalized spectra. *Zeitschrift fur Mathematische Logik und Grund. der Math.*, 21:89–96, 1975.

[FGT93] G. Fayolle, S. Grumbach, and C. Tollu. Asymptotic probabilities of languages with generalized quantifiers. In *Proc. IEEE Symp. of Logic in Computer Science*, pages 199–207, Montreal, June 1993.

[Fis87] D.H. Fishman. et al. Iris: An object oriented database managment system. In *ACM Trans. Office Information Systems, 5:1*, 1987.

[Fra54] R. Fraïssé. Sur les classifications des systèmes de relations. *Publ. Sci. Univ Alger*, I:1, 1954.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical System Theory*, 17:13–27, 1984.

[GG92] M. Gyssens and D. Van Gucht. The powerset algebra as a result of adding programming constructs to the nested relational algebra. *Journal of Computer and System Sciences*, ???, 1992.

[GM93] S. Grumbach and T. Milo. Towards tractable algebras for bags. In *Proc. 12th ACM Symp. on Principles of Database Systems*, pages 49–58, Washington, May 1993.

[GM95] S. Grumbach and T. Milo. An algebra for pomsets. In G. Gottlob and M. Vardi, editors, *Proc. Int. Conf. on Database Theory*, pages 191–207, Prague, 1995. Springer Verlag. LNCS 893.

[GMK93] S. Grumbach, T. Milo, and Y. Kornatzky. Calculi for bags and their complexity. In *4th Int. Workshop on Database Programming Languages*, New-York City, Septembre 1993. Morgan Kaufmann.

[GO93] E. Grädel and M. Otto. Inductive definability with counting on finite structures. In *Proc. of Computer Science Logic 92*, pages 231–247. LNCS, n702, 1993.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[GT95] S. Grumbach and C. Tollu. On the expressive power of counting. *Theoretical Computer Science, B*, 147, Sept. 1995. Invited to a special issue, Also Inria Research Report, n 2330, Sept 1994.

[GV90] S. Grumbach and V. Vianu. Playing games with objects. In *Proc. Int. Conf. on Database Theory*, pages 25–38, Paris, dec 1990.

[GV95] S. Grumbach and V. Vianu. Tractable query languages for complex object databases. *Journal of Computer and System Sciences*, 1995. Invited to a special issue, to appear. Extended abstract in Proc. 10th ACM Symp. on Principles of Database Systems, 1991.

[HM81] M. Hammer and D. McLeod. Database Description with SDM: a Semantic Database Model. *ACM trans. on Database Systems 6,3*, 1981.

[HS89] R. Hull and J. Su. Untyped Sets, Invention and Computable Queries. In *Proc. 8th ACM Symp. on Principles of Database Systems*, 1989.

[HS91] R. Hull and J. Su. On the expressive power of database queries with intermediate types. *Journal of Computer and System Sciences*, 43(1):219–267, August 1991.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[IL90] N. Immerman and E. Lander. *Complexity Theory Retrospective*, chapter Describing Graphs : A First-Order Approach to Graph Canonization, pages 59–81. Springer Verlag, Ed. A. Selman, 1990.

[KG85] A. Klausner and N. Goodman. Multirelations semantics and languages. In *Proc. 11th Intl. Conf. on Very Large Databases, Stockholm, Sweden*, 1985.

[KV84] G.M. Kuper and M.Y. Vardi. A new approach to database logic. In *Proc. 3rd ACM Symp. on Principles of Database Systems*, 1984.

[KV88] G.M. Kuper and M.Y. Vardi. On the complexity of queries in the logical data model. In *Proc. Int. Conf. on Database Theory*, pages 267–280, Bruges, 1988.

[KV92] P. Kolaitis and M.Y. Vardi. Fixpoint logic vs. infinitary logic in finite model theory. In *Proc. 7th Symp. of Logic in Computer Science*, pages 46–57, 1992.

[LW93a] L. Libkin and L. Wong. Aggregate functions, conservative extensions, and linear orders. In *Proc. 4th Int. Workshop on database programming languages*, 1993.

[LW93b] L. Libkin and L. Wong. Some properties of query languages for bags. In *Proc. 4th Int. Workshop on database programming languages*, 1993.

[LW94] L. Libkin and L. Wong. New techniques for studying set languages, bag languages and aggregate functions. In *Proc. 13th ACM Symp. on Principles of Database Systems*, 1994.

[MD86] F. Manola and U. Dayal. Pdm: An objected-oriented data model. In *Proc. Intl. Workshop on Object Oriented Database Systems, Asilomor,CA*, 1986.

[Mum90] I.S. Mumick. et al. The magic of duplicates and aggregates. In *Proc. 16th Intl. Conf. on Very Large Databases, Brisbane, Australia*, 1990.

[PG88] J. Paredaens and D. Van Gucht. Possibilities and limitations of using flat operators in nested algebra expressions. In *Proc. 7th ACM Symp. on Principles of Database Systems*, pages 29–38, 1988.

[PG92] J. Paredaens and D. Van Gucht. Possibilities and limitations of using flat operators in nested algebra expressions. *ACM Trans. on Database Systems*, 1992.

[Suc93] D. Suciu. Bounded fixpoints for complex objects. In *4th Int. Workshop on Database Programming Languages*, New-York City, Septembre 1993. Morgan Kaufmann.

[TF86] S.J. Thomas and P. C. Fischer. Nested relational structures. In P. C. Kanellakis and F. Preparata, editors, *Advances in Computing Research, vol.3*, pages 269–307. JAI Press, 1986.

[Ull88] J.D. Ullman. *Database and Knowledge Base Systems.* Computer Science Press, 1988.

[Won93] L. Wong. Normal form and conservative properties for query languages. In *Proc. 12th ACM Symp. on Principles of Database Systems*, 1993.