Database Theory Column

Victor Vianu
U.C. San Diego
vianu@cs.ucsd.edu

One of the characteristics of advanced database systems is that they must handle data which no longer fits the traditional relational style. The present column summarizes recent results on one such extension, where databases handle *bags* (multisets) rather than just sets. The results illustrate an important research direction in Database Theory.

Query Languages for Bags
Expressive Power and Complexity[1]

Stéphane Grumbach	**Leonid Libkin**	**Tova Milo**	**Limsoon Wong**
INRIA	Bell Laboratories	Tel Aviv University	ISS
stephane.grumbach@inria.fr	libkin@bell-labs.com	milo@math.tau.ac.il	limsoon@iss.nus.sg

Abstract

Most database theory focused on investigating databases containing sets of tuples. In practice databases often implement relations using bags, i.e. sets with duplicates. In this paper we study how database query languages are affected by the use of duplicates. We consider query languages that are simple extensions of the (nested) relational algebra, and investigate their resulting expressive power and complexity.

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 [Fis87, HM81] 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 [Alb91, Klu82, OOM87], 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, it is only recently that the expressive power of languages for manipulating bags has been investigated by the authors of the present paper [GM93, GMK93, LW93, LW93a, LW94]. We give here a summary of the main results on the expressive power and complexity of bag languages. We address the following issues:

(1) Design of a language for bags, BALG, playing a role similar to that of the relational algebra, RA, for sets; (2) Relative expressive power of the primitives of the bag algebra; (3) Relationship between set and bag languages; (4) Complexity of bag languages; and (5) Limitations of expressive power of the basic bag language.

[1] Appeared in *ACM SIGACT News*, 27(2):30–37, June 1996.

A brief summary of the results of this paper is given below.

- BALG has more expressive power than RA.
- Some properties enjoyed by RA do not hold for BALG. For example, BALG does not admit 0/1 laws, and can express some queries that do not have AC0 complexity.
- BALG is equivalent in expressive power to RA with arithmetic and aggregate functions.
- BALG has LOGSPACE complexity.
- Even though BALG has more power than RA, it cannot express recursive queries such as transitive closure and connectivity test.

2 An Algebra for Bags

The algebra presented here extends the complex object algebra [AB95] in the spirit of the bag algebras of [Alb91]. To give motivation for the operations in this language, we use the approach combining [Car88] and [BBW92]. The idea is that a data-oriented language must be organized around the type system of its data objects. For each type constructor, we need two kinds of operations. The *introduction* operations build objects of a given type. The *manipulation* operations compute over objects of a given type. We also need operations that provide *interaction* between type constructors. We present below the basic operations for bags and records, following an extension of [BBW92] to bags.

We assume the existence of a number of basic types $b_1, b_2, \ldots$, such as Booleans, integers, and strings. Types are defined using the basic types, and the tuple and bag constructors. $[T_1, ..., T_n]$ is a *tuple* type, whose domain is the set of tuples over $T_1, \ldots, T_n$. That is, $dom([T_1, ..., T_n]) = dom(T_1) \times \ldots \times dom(T_n)$. A bag is a (homogeneous) collection of objects that may contain duplicates. $\{\!|T|\!\}$ is a *bag* type, whose domain is the set of finite 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.

We assume that all the operations are typed in a polymorphic way. The restrictions on the input types of operations assure that the output is a homogeneous bag. For example, additive bag union ($\uplus$) can only be applied on bags of the same type. The type system is obvious and we omit the formal definitions. The reader can find them in [GM93, LW93, LW94]. In the presentation below, we use one level of lambda-abstraction ($\lambda x.e(x)$, where x ranges over objects of a given type) and conditional *if* $c(x)$ *then* $f(x)$ *else* $g(x)$, where c is of type $T \rightarrow bool$ and both f and g are of type $T \rightarrow T'$. As explained in [BBW92], adding these constructs does not increase expressiveness. On the other hand, it allows us to express certain operations in a simpler way.

Operations of the Basic Bag Language (BBL**)**

- Operations on records
  - Introduction operation: *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.
  - Manipulation operation: *Attribute projections* (α_i): $\alpha_i([o_1, ..., o_n]) = o_i$.

- Operations on bags
  - Introduction operations:

    Empty bag: We use the $\{\!|\,|\!\}$ constant to denote the empty bag.

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

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 $n = p + q$.

  – Manipulation operation:

  extension (EXT): if f is a function of type $T \rightarrow \{\!|T'|\!\}$, then EXT_f extends f to a function of type $\{\!|T|\!\} \rightarrow \{\!|T'|\!\}$ by $\mathrm{EXT}_f(\{\!|x_1, \ldots, x_n|\!\}) = f(x_1) \uplus \ldots \uplus f(x_n)$.

  We use MAP_g as a syntactic sugar for $\mathrm{EXT}_{\beta \circ g}$.

- Interaction operation: *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 $n = pq$.

The operations defined so far constitute our **basic bag language**, BBL. This language does not contain a number of algebraic operations such as difference or duplicate elimination. We consider adding them to BBL and then describe their expressive power relative to BBL.

Additional operations on bags

- *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 $n = inf(p, q)$.

- *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.

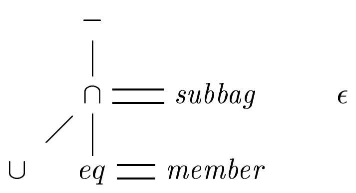
- *Equality test*, *eq*: *eq* has type $T \times T \rightarrow bool$. $eq(o, o')$ is *true* iff o and o' are equal objects.

- *Membership test*, *member* of type $T \times \{\!|T|\!\} \rightarrow bool$ returns true on a pair (o, B) iff o p-belongs to B for $p > 0$.

- *Subbag test*, *subbag* of type $\{\!|T|\!\} \times \{\!|T|\!\} \rightarrow bool$ returns true on a pair (B, B') iff whenever o p-belongs to B, then o p'-belongs to B' for some $p' \geq p$.

Do we need to add *all* these operations to BBL to get a standard bag algebra? Some operations are interdefinable, e.g. *member* and *subbag* tests are expressible using BBL and bag difference. The following characterizes precisely the relative expressive power of the additional operations.

Theorem 2.1 *With respect to* BBL, *the expressive power of these additional operations is as follows:* $-$ *can express all primitives other than* ϵ. ϵ *is independent of the rest of the primitives.* $\cap$ *is equivalent to subbag and can express both* $\cup$ *and eq. member and eq are interdefinable, both are independent of* $\cup$, *and together with* $\cup$ *can express* $\cap$. $\square$



Thus, as our **standard bag algebra** BALG, we take BBL endowed with the strongest combination of primitives, that is, $-$ and ϵ. (This language was called BQL — bag query language — in [LW93a, LW94].) Note that the operations above work for *flat* bags (bags of records with attributes of basic types) as well as for *nested* bags (where tuple attributes can also contain nested bags).

The bag algebra can express many operations commonly found in database languages. For instance, $\mathrm{MAP}_{\lambda x.[\alpha_2(x),\alpha_3(x)]}$ denotes the projection of a tuple type on its second and third arguments. (For brevity, we shall denote below the map projecting the attributes $i_1, \ldots, i_n$ by $\pi_{i_1,\ldots,i_n}$). More interestingly, bag manipulation offers *gain* of expressive power. It allows the definition of several fundamental database primitives. For example, bags can be used to simulate aggregate functions, such as *sum* and *count*. For this, an integer i can be represented by a bag containing i occurrences of an element, say a, and if B is a bag of tuples, then $count(B) = \pi_1(\{\![a]\!\} \times B)$.

3 Bag Languages vs Set Languages

As in the classical relational case, we are aiming for characterizations of the expressiveness of BALG in terms of complexity classes of queries. In particular, we compare the expressive power of the bag algebra to that of the relational algebra, RA. Since we are considering complex (nested) objects as well as flat relations, we also look at the relationship between BALG when applied on complex objects and the *nested* relational algebra, NRA [AFS89]. NRA is an extension of RA to nested relations. Essentially, it is the same as BALG with complex objects, when all operations on bags are turned into their set analogs. That is, $-$ becomes the usual set difference, both $\cup$ and $\uplus$ become the usual set union, ϵ becomes the identity function and so on. It was shown that that nesting does not add any extra power to RA in the sense that any NRA query from flat relations to flat relations can be defined in RA [Won93].

We first look at the primitives one needs to add to RA or NRA to match the corresponding language on bags. Let *arithmetic* stands for the following addition to the language. It includes the type *nat* of natural numbers, together with the operations of addition, multiplication, and modified subtraction $\dot{-}$ (i.e. $n \dot{-} m = \max(0, n-m)$) and a general summation operator Σ_f. Here f is of type $T \to nat$, and $\sum_f$ is of type $\{T\} \to nat$ with the semantics $\sum_f(\{x_1, \ldots, x_k\}) = f(x_1) + \ldots + f(x_k)$.

Theorem 3.1 BALG *when restricted to flat bags is equivalent to* RA + *arithmetic.* BALG *over nested bags is equivalent to* NRA + *arithmetic.* $\Box$

We next present an example illustrating the power of the bag difference.

Example 3.1 Consider a directed graph whose edges are recorded in a binary relation G. The query $(\pi_2(\sigma_{2=a}G)) - (\pi_1(\sigma_{1=a}G)) \neq \emptyset$ expresses the fact that the in-degree of a node a is bigger than its out-degree. Here $\sigma_{i=a}$, 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", 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.

Another area where BALG differs from RA is its behavior with respect to asymptotic probabilities of definable properties. Consider unnested databases. The probability, $\mu_n(P)$, that a (boolean) property P holds for databases over an n-element domain is the ratio of the number of databases over an n-element domain satisfying P to the number of all databases over an n-element domain. The asymptotic probability of P is the limit of this ratio (if it exists) when n goes to ∞. Boolean expressions in RA containing no constants admit a 0/1 law (that is, the asymptotic probability exists and can only be 0 or 1), while BALG doesn't enjoy such a regularity.

Consider a schema over two monadic relation symbols R and S. The query $(\pi_1(R \times R) - \pi_1(R \times S)) \neq \emptyset$ expresses the fact that the cardinality of R is bigger than the cardinality of S. 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. For more details on the asymptotic probabilities of queries expressing counting properties, see [GT95, FGT93].

4 Complexity of BALG

BALG differs from RA not only in expressive power but also in its data complexity. Indeed BALG does not enjoy the AC0 data complexity upper-bound of RA. 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 AC0 upper-bound offers potential for efficient parallel evaluation. RA enjoys an AC0 upper-bound [AHV94], and so does NRA [ST94]. It is well known that there are simple functions that are not computable in AC0, such as multiplication and parity test [FSS84]. It follows then from Theorem 3.1, that BALG is not in AC0.

As a more interesting example of violation of the AC0 upper bound, we show that the *parity* of the cardinality of a relation (bag with no duplicates) becomes definable in BALG in the presence of an *order* on the domain. The following boolean expression states that the parity of the cardinality of relation R is even: $\sigma_{\lambda x.(\mathrm{MAP}_{[a]}(\sigma_{\lambda y.(y \leq x)} R) = \mathrm{MAP}_{[a]}(\sigma_{\lambda y.(x<y)} R))}(R) \neq \{\!|\,|\!\}$.

The expression 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 tuple for each element). It is clear that the existence of such an element in R guarantees parity of the cardinality of R.

Even though BALG is not contained in AC0, its complexity is nevertheless not too dramatic.

Theorem 4.1 BALG $\subset$ LOGSPACE.

5 Limitations of expressive power of BALG

Although BALG has more expressive power than RA (equal cardinality is definable), some fundamental limitations on the expressive power of first-order logic still hold. We first consider bags containing occurrences of a single constant c. The query *bag-even* tests the parity of the number of duplicates in a bag. More precisely, *bag-even*$(B) =$ *true* if c's multiplicity in B is even, and *false* otherwise.

Proposition 5.1 The query *bag-even* is not expressible in BALG.

This proposition is true for both flat and nested bags. It can be better understood if one looks at RA + *arithmetic* and NRA + *arithmetic* — the set-based languages equivalent to BALG. In this context it says that it is impossible to test if a given number n is even. The technique used to prove this proposition reduces the problem to finding roots of a polynomial, and is based on the fact that there are finitely many of those for non-zero polynomials. In fact, a more general result can be proved : Every property of the number of duplicates of a single constant that can be tested in BALG is either finite (i.e holds for a finite number of bags) or co-finite (i.e its complement holds for a finite number of bags).

It is interesting to contrast this result with definability of parity of the cardinality of a relation in the presence of order (demonstrated in the previous section). Note that RA cannot test parity even in the presence of order.

To present a more powerful inexpressibility result, assume a base type b with a countably infinite domain of uninterpreted constants. That is, only equality test is available, and no order relation is given. A graph G, that is, an object of type $\{\!|b \times b|\!\}$ without duplicates, is called a **k-multi-cycle** if it consists of a number of unconnected simple cycles of equal length $l \geq k$.

Theorem 5.2 *Let $q : \{\!|b \times b|\!\} \rightarrow bool$ be a Boolean query in* BALG. *Then there exists a number k such that $q(G) = q(G')$ for any two k-multi-cycles G and G'.*

That is, multi-cycles cannot be distinguished by BALG queries as long as their components are long enough. From this we conclude that many recursive queries are not definable in BALG, most notably those complete for LOGSPACE and NLOGSPACE.

Corollary 5.3 *The following are not definable in* BALG *over unordered domains: the parity of the cardinality of a relation, transitive closure, deterministic transitive closure, testing acyclicity, testing connectivity, testing for balanced binary trees.*

6 Extending the Language

Since BALG has a number of limitations similar to those of the relational algebra, several attempts have been made to extend the power of BALG. First, in the nested case, the *powerset* operator can be added to BALG, following the ideas of [AB95]. When applied to a bag B, *powerset* returns the bags of all subbags of B, each with multiplicity 1. A related operator, *powerbag* returns the bag of subbags of B, in which the multiplicity of each subbag is the product of multiplicities of its elements in B. It was shown in [LW93a] that *powerset* and *powerbag* are interdefinable. The complexity of BALG+*powerset* and BALG+*powerbag* was studied in [GM93].

However, there are serious problems with the *powerset*. While all queries in Corollary 5.3 become definable, they are not definable efficiently. That is, under the natural evaluation strategy, computing transitive closure would require exponential space [SP94]. Even though more advanced query evaluation techniques proposed in [AH95, Lib95a] reduce this to polynomial space, it is unknown whether the exponential time complexity bound can be improved.

Another way of enriching the language is adding the structural recursion operator to it. This was done for sets in [BBN91], and extended to bags in [LW93a]. Without getting into details, we only remark here that the main problem with using structural recursion is that it requires certain preconditions on its parameters for well-definedness. However, these preconditions are generally undecidable [BS91]. In order to overcome this, [LW93a] introduced the loop construct *loop* that takes as a parameter a function of type $T \rightarrow T$, an object o of type T and a bag B of any type $\{\!|T'|\!\}$, and returns f applied to o $card(B)$ times. It was shown in [LW93a] that nested BALG with structural recursion is equivalent to nested BALG with *loop*.

As a conclusion of this section, we present one result on expressiveness of these new primitives. Recall that a natural number n can be simulated as a bag with n occurrences of a constant. Thus, we can speak of classes of arithmetic functions simulated by bag languages.

Proposition 6.1 *The classes of arithmetic functions simulated by nested* BALG *with powerset and loop are the classes of Kalmar-elementary and primitive-recursive functions, respectively.*

7 Open problems

There are two open problems we would like to mention in conclusion. First, 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 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 have not investigated this issue. Nevertheless, an extension of the relational machines with counters was proposed in [GO93], and we have seen that there is a close relationship between bags and counters.

Second, we conjecture that the bounded degree property of [LW94] extends from RA to BALG. This property says that for any graph query q and a number k, one can find a number $c(q,k)$ such that for any graph G whose in- and out-degrees are bounded by k, the graph $q(G)$ does not have more than $c(q,k)$ distinct in- and out-degrees. This property, which holds in RA, allows us to prove a number of inexpressibility results easily and in a uniform way. For instance, inexpressibility of most queries from Corollary 5.3 would follow immediately.

References

[AB95] S. Abiteboul and C. Beeri. The power of languages for the manipulation of complex values. *The VLDB Journal* 4(1995), 727–794.

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

[AH95] S. Abiteboul and G. Hillebrand. Space usage in functional query languages. In *Proc. of Intl. Conf. on Database Theory*, pages 437–454. LNCS 893, Springer Verlag, 1995.

[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*, 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.

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

[BBW92] 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.

[Car88] L. Cardelli. Types for data-oriented languages. In *Proceedings of EDBT-88* (J.W. Schmidt, S. Ceri and M. Missikoff eds), LNCS 303, Springer Verlag, 1988.

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

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

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

[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. Full version to appear in *Journal of Computer and System Science.*

[GMK93] S. Grumbach, T. Milo, and Y. Kornatzky. Calculi for bags and their complexity. In *4th Int. Workshop on Database Programming Languages*, September 1993, Springer Verlag, 1994.

[GO93] E. Grädel and M. Otto. Inductive definability with counting on finite structures. In *Proc. of Computer Science Logic 92*, pages 231–247. LNCS 702, 1993.

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

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

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

[Klu82] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM* 29 (1982), 699–717.

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

[Lib95a] L. Libkin. Normalizing incomplete databases, In *Proc. 14th ACM Symp. on Principles of Database Systems, San Jose CA*, May 1995, pages 219–230.

[LW93] L. Libkin and L. Wong. Aggregate functions, conservative extensions, and linear orders. In *Proc. 4th Int. Workshop on Database Programming Languages*, September 1993, Springer Verlag, 1994.

[LW93a] L. Libkin and L. Wong. Some properties of query languages for bags. In *Proc. 4th Int. Workshop on Database Programming Languages*, September 1993, Springer Verlag, 1994.

[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, pages 155–166.

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

[OOM87] G. Ozsoyoglu, Z. M. Ozsoyoglu, and V. Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM TODS*, 12 (1987), 566–592.

[SP94] D. Suciu and J. Paredaens. Any algorithm in the complex object algebra with powerset needs exponential space to compute transitive closure. In *Proc. 13th ACM Symp. on Principles of Database Systems*, 1994.

[ST94] D. Suciu and V. Tannen. A query language for NC. In *Proc. 13th ACM Symp. on Principles of Database Systems*, 1994.

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