# The expressive power of cardinality-bounded set values in object-based data models*

Jan Van den Bussche† Dirk Van Gucht‡

May 18, 1994

**Abstract**

In object-based data models, complex values such as tuples or sets have no special status and are represented just as any other object. However, different objects may represent the same value, i.e., duplicates may occur. It is known that typical object-based models supporting first-order queries, standard object creation, and while-loops, cannot in general guarantee the duplicate-freeness of representations of set values. In this paper, we consider a number of extensions of the basic object-based model which provide exactly this ability, under the assumption that a fixed bound is known on the cardinality of the set values. We show that these extensions are all equivalent to each other. Our main result is that increasing the cardinality bound from $m$ to $m+1$ yields strictly more expressive extensions, except for $m=0$ and $m=3$. We thus establish a non-collapsing hierarchy.

*Extended and revised version of a paper presented at the Fourth International Conference on Database Theory [21].

†Research Assistant of the NFWO. Address: Dept. Math. & Computer Sci., University of Antwerp (UIA), Universiteitsplein 1, B-2610 Antwerp, Belgium. E-mail: vdbuss@wins.uia.ac.be.

‡Computer Science Department, Indiana University, Bloomington, IN 47405-4101, USA. E-mail: vgucht@cs.indiana.edu.

# 1 Introduction

In the past decade, there has been a lot of interest in accommodating more complex data structures as first-class citizens in database systems, a feature not offered by standard relational systems. Recent work in this field lead to the definition of two new data models: the *complex value* model and the *object-based* model. (There were also proposals to combine the two approaches [2, 5, 10, 17].)

The complex value model[1] [1] is an extension of the standard relational model. While the relational model offers collections of tuples, the complex value model offers collections of arbitrary combinations of sets and tuples called *complex values*. In the object-based model [8, 11, 13, 16, 15], a database is essentially thought of as a labeled graph of objects, where each set of equally labeled objects comprises a so-called class. The edges between objects in the graph express properties and are labeled by property names. This approach is inspired by the object-oriented philosophy [14], but can in fact be traced back to the Functional Data Model [18].

A difference between the complex value approach and the object-based approach is that in the latter, set values are not explicitly part of the data model. The usual way of representing a set in such a model is by having an object $o$, with equally labeled edges linking $o$ to each element of the set. A class of objects then represents a collection of set values if each object in the class represents a set in the collection, and vice versa, each set in the collection is represented by an object in the class. However, it may occur that two different objects in the class represent the same set, i.e., *duplicates* may occur.

In this paper, we will focus on object-based queries whose result is a collection of set values. Such queries augment the database with the new objects and edges necessary for representing the desired collection. The fundamental query language for relational databases, the relational calculus, can be adapted for this purpose. More specifically, a relational calculus query over the database graph can be used for object creation by creating a new object for each tuple in its result. And if the result is a binary relation, it can be alternatively used for edge addition. A simple yet powerful object-based query language, which we call OBQL, can thus be obtained by providing

[1] Also known as the complex object, nested relational, $NF^2$, or unnormalized model.

object creation and edge addition as basic statements and closing off under composition and while-loops. This language subsumes many object-based query languages proposed in the literature.

However, it is known [20] that if we insist on duplicate-free representations, there are very simple collections of set values that are inexpressible in OBQL. For example, the query asking for all subsets of two elements of a given class is not expressible without duplicates.

Duplicate-free representations have a number of apparent practical advantages. Obviously, duplicates cause redundancy in the database. Another advantage concerns the efficient answering of queries involving the equality of set values. In arbitrary representations, checking for equality of two sets requires an expensive comparison of all elements. However, if every set value is represented by a unique object, checking equality amounts to one single comparison of the corresponding object identifiers. A third advantage is efficiency of representation. If a client program asks for a collection of set values, it is useful if the server program can deliver the collection in the form of a unique handle to each set. Different handles to the same set value, i.e., duplicates, would be very undesirable in this situation.

Hence, it is desirable to enrich OBQL with an additional primitive for the creation of duplicate-free set representations. The obvious candidate for this, considered in [15], is an explicit *powerset* operation. An alternative, considered in [8], is the *abstraction* operation, which provides a quotient construction, creating a unique representative for each equivalence class of duplicate objects. These two options for enriching OBQL are equivalent [20].

In the present paper, we are motivated by the observation that in many practical applications, the set values appearing in the database have *bounded cardinality*: their cardinality is often known to be bounded by a fixed natural number $m$. For example, we might know in advance that each student will take at most ten courses. Or, in a genealogy database, any person has at most four grandparents. We define a natural cardinality-bounded restriction of the abstraction operation, and show that it is equivalent to various cardinality-bounded, duplicate-free set-object creation operations. In particular, the above-mentioned equivalence between abstraction and powerset holds also under the restriction of bounded cardinality. Our second and main result is that enriching OBQL with $m$-bounded abstraction yields a strictly less expressive extension than the one obtained by enriching with $m+1$-bounded abstraction (except for $m = 3$). We thus reveil a non-collapsing hierarchy

of duplicate-free set creation operations in object-based data models, the supremum of which is general abstraction.

The proofs of our results carry some interest on their own, and provide more insight on the issues of object creation, set value representation, and duplicates. For instance, we show that object identifiers can be interpreted as hereditarily finite sets. Based on this insight, we reduce the ability to do duplicate-free $m$-bounded set creation to the existence of a certain type of hereditarily finite set that is fixed by all permutations of $\{1, \ldots, m\}$. After this reduction, our hierarchy result then follows from some basic facts of group theory.

The hierarchy we establish in this paper should be contrasted with other hierarchies established in the context of data manipulation languages for complex objects [9, 12]. These hierarchies are based on the nesting depth of sets, while our hierarchy is based on the cardinality of sets. At the end of the paper, we mention a few interesting problems which remain open.

# 2 The object-based data model

In this section, we define a general object-based data model, which serves as a formal framework capturing the features (relevant to this paper) of many object-oriented database systems encountered in practice. Our formalism, which views database schemes and instances as directed, labeled graphs, is close to that of the earlier proposals LDM [15] and GOOD [8]. We will define database schemes and instances as special kinds of graphs, and we will introduce a simple yet powerful object-based query language, called OBQL.

It is customary in object-based models to depict a database scheme as a graph. So we assume the existence of infinitely enumerable sets of *class names* and *property names*, and define:

**Definition 2.1** *A* scheme *is a finite, edge-labeled, directed graph. The nodes of the graph are class names and the edges are triples $(B, e, C)$, where $B$ and $C$ are nodes and the edge label $e$ is a property name.*

A database instance can now be defined as a graph consisting of objects and property-links, whose structure is constrained by some database scheme. So we assume the existence of an infinite supply of *objects*, and define, for an arbitrary scheme $S$:

**Definition 2.2** *An* instance *over $S$ is a finite, labeled, directed graph. The nodes of the graph are objects. Each node $o$ is labeled by a class name $\lambda(o)$ of $S$. The edges are triples $(o, e, p)$, where $o$ and $p$ are nodes and the edge label $e$ is a property name of $S$ such that $(\lambda(o), e, \lambda(p))$ is an edge of $S$.*

The set of all objects in an instance labeled by the same class name $C$ will be called *the class $C$*.

Before turning to the object-based query language OBQL, we must first specify what we mean with the notion of query in the object-based data model. In the relational model, a query is typically considered as a function, mapping an input database to an output relation. This output relation is often materialized as derived information, or used as part of the input to a subsequent query. Hence, it is natural to view a query alternatively as a function which augments an input database with a new, derived relation. This view of a query can be readily adopted in the object-based data model: a query is a function which augments an input instance with new objects and edges. Correspondingly, OBQL provides two basic operations, one for object creation and one for edge addition.

Object creation and edge addition are based on the following adaptation of the relational calculus to object databases. With a scheme $S$, we can associate a standard, first-order, many-sorted logic. The class names of $S$ are the sorts, and for each edge $(B, e, C)$ in the scheme there is a binary, sorted predicate name $e(B, C)$. Given an instance $I$ over $S$, a sort $C$ is interpreted by the class $C$ in $I$, and the predicate $e(B, C)$ is interpreted by the set of all $e$-labeled edges going from objects of class $B$ to objects of class $C$. Now let $\Phi(x_1, \ldots, x_n)$ be a formula over $S$, and let $C_i$ be the sort of $x_i$. Evaluating $\{(x_1, \ldots, x_n) \mid \Phi\}$ over $I$ yields an $n$-ary relation consisting of all tuples $(o_1, \ldots, o_n)$ of objects in $I$ satisfying $\Phi$. Note that $o_i$ will be in class $C_i$.

The *object creation* operation:

$$\Gamma \equiv C[e_1 : x_1, \ldots, e_n : x_n] \Leftarrow \Phi$$

provides a natural way to augment the database with a representation of the above $n$-ary relation. Here, $C$ is some class name and $e_1, \ldots, e_n$ are property names. The effect of $\Gamma$ on schemes $S$ and instances $I$ is formally defined as follows:

**Definition 2.3**

- $\Gamma(S)$ *is the scheme obtained by augmenting*[2] $S$ *with node* $C$ *and edges* $(C, e_i, C_i)$*, for* $i = 1, \ldots, n$*.*
- $\Gamma(I)$ *is the instance over* $\Gamma(S)$ *obtained from* $I$ *by adding, for each tuple* $(o_1, \ldots, o_n)$ *of objects in* $I$ *such that* $\Phi(o_1, \ldots, o_n)$ *is true in* $I$*, a new object* $o$ *with label* $C$*, together with edges* $(o, e_i, o_i)$*, for* $i = 1, \ldots, n$*.*

If $n = 2$, then evaluating the formula $\Phi$ yields a binary relation, which can be used not only for object creation, but also for edge addition. Indeed, each pair in the relation can be interpreted as a set of derived edges. These can be added to the database using the *edge addition* operation

$$\Delta \equiv e(x_1, x_2) \Leftarrow \Phi,$$

where $e$ is some property name. The effect of $\Delta$ on schemes and instances is formally defined as follows:

**Definition 2.4**

- $\Delta(S)$ *is the scheme obtained by augmenting* $S$ *with the edge* $(C_1, e, C_2)$*.*
- $\Delta(I)$ *is the instance over* $\Delta(S)$ *obtained by augmenting* $I$ *with an edge* $(o_1, e, o_2)$ *for each tuple* $(o_1, o_2)$ *of objects in* $I$ *such that* $\Phi(o_1, o_2)$ *is true in* $I$*.*

Queries can now be expressed in OBQL by means of arbitrary compositions of object creation and edge addition operations. Furthermore, these compositions can be iterated using a while-loop construct of the form

**while change do** $op_1; \ldots; op_k$ **od**.

The body of the loop is executed as long as the instance under operation changes (which might be forever.)

We conclude this section with some remarks on some specific features of OBQL.

[2]By *augmenting* a graph $G$ with a node or edge $x$, we mean adding $x$ to $G$ provided $x$ does not already belong to $G$.

We allow that the labels of objects and edges that are added to an instance by an object creation or edge addition already exist in the scheme of that instance. This provision is necessary for adding derived information incrementally, e.g., using a while-loop. For example, the following program computes the transitive closure of a database graph whose objects are all in the same class and whose edges all have the same label $e$. The edges of the transitive closure will get the label $e^*$.

$e^*(x, y) \Leftarrow e(x, y)$;
**while change do** $e^*(x, y) \Leftarrow \exists z : e(x, z) \wedge e^*(z, y)$ **od**

A program expressing a query will often create a lot of auxiliary objects and edges that are only used for storing temporary results in the course of the computation, and should be omitted from the end result. We will not define this practice formally; it will always be clear from the context which of the labels are only temporary.

A number of object-based data models considered in the literature [8, 13, 16] use an alternative semantics for object creation, which we will call *weak* semantics, and which is often natural and useful. Recall Definition 2.3 of the object creation operation. The weak variant of this operation, written

$$C[e_1 : x_1, \ldots, e_n : x_n] \Leftarrow_{\text{weak}} \Phi,$$

only adds a new object $o$ (as specified in the definition) if there is not already a $C$-labeled object $o'$ with edges $(o', e_i, o_i)$ in the database. Hence, it is equivalent to

$$C[e_1 : x_1, \ldots, e_n : x_n] \Leftarrow \Phi \wedge \neg\exists x : e_1(x, x_1) \wedge \cdots \wedge e_n(x, x_n).$$

Thus, the weak semantics can be simulated in our semantics; actually, the converse is true as well. The converse simulation uses an auxiliary class $T$, structured as an ever-growing stack. The stack is initialized with a bottom object using the "zero-ary" object addition:

$$T[\,] \Leftarrow_{\text{weak}} \textbf{true}.$$

The object creation

$$C[e_1 : x_1, \ldots, e_n : x_n] \Leftarrow \Phi$$

is then simulated by first pushing a new object on the stack:

$$T[prev : t] \Leftarrow_{\text{weak}} \neg\exists t' : prev(t', t).$$

(Here, $t$ and $t'$ are variables of sort $T$; the formula states that $t$ is the top of the stack.) The actual object creation is then performed by

$$C[e_1 : x_1, \ldots, e_n : x_n, e : t] \Leftarrow_{\text{weak}} \Phi \wedge \neg\exists t' : prev(t', t).$$

In words, the new object must be connected to the top of the stack with a temporary edge labeled $e$. This guarantees that it will indeed be created, regardless of whether there already exists an object with the same $e_1, \ldots, e_n$-edges, since such an object will be connected to a lower object in the stack.

# 3 Representation of set values

Complex values, such as set values, are not explicitly part of the object-based data model defined in the previous section. Instead, set values are represented by objects through their properties. Consider as an example a scheme containing class names *Student* and *Course*, with an edge from *Student* to *Course* labeled by the property name *takes*. In an instance, there will be objects labeled *Student*, i.e., students, and objects labeled *Course*, i.e., courses. Each student is connected to the courses he takes by edges labeled *takes*. We say that each student *represents* the set of courses he takes. Hence, the collection $V$ of all sets of courses represented by some student is represented by the class *Student*. However, since different students may take exactly the same courses, different students may represent the same set value: we say in this case that the representation of $V$ by class *Student* is not *duplicate-free.*

Of course, we do not want to disallow "duplicate" students. Nevertheless, it might be desirable to also have a representation of $V$ which is duplicate-free. Some advantages of duplicate-free representations have been pointed out in Section 1; we will now illustrate one of them. Assume we have an additional class, say *Set*, representing the collection $V$ without duplicates. So, in the scheme, there is an additional edge from *Set* to *Course* labeled with *contains*, say. In the instance, each *Set*-object is linked via *contains*-edges to precisely the courses of a set in $V$. All sets in $V$ are represented

in this way, and no two *Set*-objects represent the same set. We can then derive new edges, labeled *set_courses*, from these *Set*-objects to students by the following edge addition operation:

$$set\_courses(z, s) \Leftarrow \forall c : takes(s, c) \leftrightarrow contains(z, c)$$

(Here, $s$, $z$ and $c$ are variables of sort *Student*, *Set* and *Course*, respectively.) After this operation, each student is linked to the unique *Set*-object representing the set of courses taken by that student. Note also that the *contains*-edges have now become dispensable, since they can be recovered by first following a *set_courses*-edge, then a *takes*-edge. After the preprocessing performed by this edge addition, queries concerning the equality of sets of courses can now be answered very efficiently. To test whether students take exactly the same courses, we can simply check whether they are linked to the same *Set* object (by a *set_courses*-edge).

A natural question now is the following: can we generate this duplicate-free class *Set* (together with the *set_courses*-edges) by means of an OBQL query? This question can be put more generally in terms of the *abstraction* operation, introduced in [8]. The abstraction is an operation for turning an arbitrary given representation into a duplicate-free one. More specifically, given a class $C$, it creates for each equivalence class $Z$ of duplicate objects in $C$ (with respect to some property $p$) a unique representative object (labeled $K$) which is linked to all members of $Z$ (by edges labeled $e$). Here, two objects are called *duplicates* with respect to $p$ if they represent the same set value with respect to $p$, i.e., if they are linked to the same set of objects by edges labeled $p$. We will write the abstraction operation as

$$\textbf{abstr}\ K[e] \Leftarrow C/p.$$

For example, we can create the desired class *Set* of the above example as follows:

$$\textbf{abstr}\ Set[set\_courses] \Leftarrow Student/takes.$$

As just defined, the abstraction operation works on all objects of a class $C$. It is often useful however to work only on a subset of the class, determined by some formula $\Phi(x)$, with $x$ a variable of sort $C$. We will write this generalized version of the abstraction operation, which we will call the *qualifying* version, as

$$\textbf{abstr}\ K[e] \Leftarrow C/p \mid \Phi.$$

E.g., in the above example, if we want only *Set*-objects for the sets of courses taken by married students, we use:

$$\textbf{abstr}\ \mathit{Set}[\mathit{set\_courses}] \Leftarrow \mathit{Student}/\mathit{takes} \mid \exists p : \mathit{spouse}(s, p)$$

(Assuming the database scheme contains an edge labeled *spouse* from *Student* to, say, class name *Person*; variables $s$ and $p$ are of sort *Student* and *Person*, respectively.)

Returning now to our question, we can rephrase it as follows: is abstraction expressible in OBQL? This question was answered negatively in [20]. It was shown there that even very simple collections of set values are inexpressible in OBQL without duplicates, such as the query asking for all subsets of two elements of a given class. The latter query is a kind of cardinality-restricted powerset construction, which can be defined in general as follows:

**Definition 3.1** *Let $C, K$ be class names and let $e$ be a property name. Let $m$ be a natural number. Then applying the $m$-*restricted powerset operation:

$$\textbf{powerset}_m\ K[e] \Leftarrow C$$

*results in the addition, for each subset $Z$ of class $C$ of cardinality $m$, of a new object $o$ with label $K$ together with edges $(o, e, o')$ to each $o' \in Z$.*

We can also define a cardinality-bounded version of the cardinality-restricted powerset operation by replacing the phrase "of cardinality $m$" in Definition 3.1 with "of cardinality at most $m$." Let us denote this operation by $\textbf{powerset}_{\leq m}$.

Like the qualifying version of the abstraction operation, a qualifying version of the cardinality-bounded powerset operation is often useful. We conceive qualifying powerset construction as an intuitive set-based dual of OBQL's standard object creation operation:

**Definition 3.2** *Let $C, K$ be class names and let $e$ be a property name. Let $m$ be a natural number. Let $\Phi(x_1, \ldots, x_k)$, with $k \leq m$, be a formula where each $x_i$ is of sort $C$. Then applying the $m$-*bounded set creation operation:

$$K[e : \{x_1, \ldots, x_k\}] \Leftarrow \Phi(x_1, \ldots, x_k)$$

*results in the addition, for each collection $\{o_1, \ldots, o_k\}$ of (not necessarily distinct) objects of class $C$ such that $\Phi(o_1, \ldots, o_k)$ is true, of a new object $o$ with label $K$ together with edges $(o, e, o_i)$ to each $o_i$.*

For example, if we know that each student takes at most ten courses, then the *Set*-objects desired in the earlier example can be added as follows:

$$Set[set\_courses : \{c_1, \ldots, c_{10}\}] \Leftarrow \exists s : takes(s, c_1) \wedge \cdots \wedge takes(s, c_{10}).$$

Inspired by all these cardinality-bounded operations, we can also define a cardinality-bounded version of the original abstraction operation (qualifying or not), which creates representative objects only for those equivalence classes of objects representing a set value of cardinality at most $m$, for some fixed natural number $m$. Let us denote this by $\mathbf{abstr}_{\leq m}$.

Given any of the new kinds of operations defined in this section, say *op*, we can extend OBQL by allowing operations of that kind as basic statements (besides object creation and edge addition.) The such extended language will be called OBQL + *op*. It turns out that all these different possible extensions are equivalent:

**Theorem 3.3** *The following languages are equivalent:*

1. OBQL + *m-bounded set creation;*
2. OBQL + *qualifying* $\mathbf{abstr}_{\leq m}$*;*
3. OBQL + $\mathbf{abstr}_{\leq m}$*;*
4. OBQL + $\mathbf{powerset}_{\leq m}$*;*
5. OBQL + $\mathbf{powerset}_m$.

**Proof.**

**(1) ⇒ (2).** Given a program in OBQL + $m$-bounded set creation, an equivalent program in OBQL + qualifying $\mathbf{abstr}_{\leq m}$ can be obtained by replacing each statement of the form

$$K[e : \{x_1, \ldots, x_k\}] \Leftarrow \Phi(x_1, \ldots, x_k) \qquad (k \leq m)$$

with the following statements. Variables $x_i$, $z'$, $z$, and $x$ are of sort $C$, $K'$, $K$, and $C$, respectively.

$K'[e' : x_1, \ldots, e' : x_k] \Leftarrow \Phi(x_1, \ldots, x_k)$;
$\mathbf{abstr}_{\leq m}\ K[e''] \Leftarrow K'/e' \mid \neg old(z', z')$;
$e(z, x) \Leftarrow \exists z' : \neg old(z', z') \wedge e''(z, z') \wedge e'(z', x)$;
$old(z', z') \Leftarrow z' = z'$

Note that at each execution of the above program fragment, new auxiliary $K'$-objects will be created, which are then discarded at the end by labeling them with *old*.[3] This reuse of the same auxiliary class name $K'$ is important if the statement simulated by the program fragment would occur in the body of a while-loop. (This technique, which is frequently used in this paper, has been used in the literature earlier, e.g., [3, Theorem 2.4.1].)

**(2) $\Rightarrow$ (3).** Given a program in OBQL + qualifying $\mathbf{abstr}_{\leq m}$, an equivalent program in OBQL + $\mathbf{abstr}_{\leq m}$ can be obtained by replacing each statement of the form

$$\mathbf{abstr}_{\leq m}\ K[e] \Leftarrow C/p \mid \Phi(x)$$

with the following statements.

$\mathbf{abstr}_{\leq m}\ K'[e'] \Leftarrow C/p$;
$K[a : z'] \Leftarrow \neg old(z', z') \wedge \exists x : e'(z', x) \wedge \Phi(x)$;
$e(z, x) \Leftarrow \exists z' : \neg old(z', z') \wedge a(z, z') \wedge e'(z', x) \wedge \Phi(x)$;
$old(z', z') \Leftarrow z' = z'$

**(3) $\Rightarrow$ (4).** Given a program in OBQL + $\mathbf{abstr}_{\leq m}$, an equivalent program in OBQL + $\mathbf{powerset}_{\leq m}$ can be obtained by replacing each statement of the form

$$\mathbf{abstr}_{\leq m}\ K[e] \Leftarrow C/p$$

with the following statements. For simplicity, we assume that in the database scheme, there is only one edge labeled $p$ leaving $C$, say $(C, p, B)$. Variable $y$ is of sort $B$.

[3] Objects are labeled *old* by attaching a loop-edge to them, labeled *old*. This explains the "binary" format of the last statement, which may seem awkward at first sight.

$\mathbf{powerset}_{\leq m}\ K'[e'] \Leftarrow B;$
$e''(z', x) \Leftarrow \neg old(z', z') \wedge \forall y : p(x, y) \leftrightarrow e'(z', y);$
$K[a : z'] \Leftarrow \neg old(z', z') \wedge \exists x : e''(z', x);$
$e(z, x) \Leftarrow \exists z' : \neg old(z', z') \wedge a(z, z') \wedge e''(z', x);$
$old(z', z') \Leftarrow z' = z'$

**(4) $\Rightarrow$ (5).** By induction on $m$. The case $m = 0$ is trivial. So assume $m > 0$. Note that a statement of the form

$$\mathbf{powerset}_{\leq m}\ K[e] \Leftarrow C$$

is equivalent to the two statements

$\mathbf{powerset}_{\leq m-1}\ K[e] \Leftarrow C;$
$\mathbf{powerset}_{m}\ K[e] \Leftarrow C$

By the induction hypothesis, it now suffices to show that given a program in OBQL + $\mathbf{powerset}_{m-1}$, there is an equivalent program in OBQL+$\mathbf{powerset}_{m}$. This can be obtained by replacing each statement of the form

$$\mathbf{powerset}_{m-1}\ K[e] \Leftarrow C$$

with the following statements. Variable $x'$ is of sort $C'$.

$C'[\ ] \Leftarrow \mathbf{true};$
$dummy(x', x') \Leftarrow \neg old(x', x');$
$C'[a : x] \Leftarrow x = x;$
$\mathbf{powerset}_{m}\ K'[e'] \Leftarrow C';$
$K[b : z'] \Leftarrow \neg old(z', z') \wedge \forall x' : e'(z', x') \rightarrow \neg old(x', x') \wedge$
$\exists x' : dummy(x', x') \wedge e'(z', x');$
$e(z, x) \Leftarrow \exists z' : \neg old(z', z') \wedge b(z, z') \wedge \exists x' : e'(z', x') \wedge a(x', x);$
$old(x', x') \Leftarrow x' = x';$
$old(z', z') \Leftarrow z' = z'$

**(5) $\Rightarrow$ (1).** Given a program in OBQL+$\mathbf{powerset}_{m}$, an equivalent program in OBQL + $m$-bounded set creation can be obtained by replacing each statement of the form

$$\mathbf{powerset}_{m}\ K[e] \Leftarrow C$$

with the statement

$$K[e : \{x_1, \ldots, x_m\}] \Leftarrow \bigwedge_{1 \leq i < j \leq m} x_i \neq x_j$$

■

# 4 A hierarchy result

In [20, 19], it was shown that the abstraction operation is not expressible in OBQL. This was done by proving that for $m \geq 2$, $\mathbf{powerset}_m$ is not expressible in OBQL. In this section, we will show:

**Theorem 4.1** *Except for* $m = 0$ *and* $m = 3$, $\mathbf{powerset}_{m+1}$ *is not expressible in* OBQL + $\mathbf{powerset}_m$.

The operations $\mathbf{powerset}_0$ and $\mathbf{powerset}_1$ are merely special cases of OBQL's standard object creation, so OBQL + $\mathbf{powerset}_{\leq 1}$ is equivalent to OBQL, settling the case $m = 0$. We next settle the case $m = 3$.

**Proposition 4.2** OBQL + $\mathbf{powerset}_3$ *is equivalent to* OBQL + $\mathbf{powerset}_4$.

**Proof.** The idea of the proof is to simulate a 4-set, say $\{1, 2, 3, 4\}$, with the 3-bounded object:

$$\Big\{ \big\{\{1,2\},\{3,4\}\big\} \ , \ \big\{\{1,3\},\{2,4\}\big\} \ , \ \big\{\{1,4\},\{2,3\}\big\} \Big\}$$

Note that this technique is ad-hoc: it does not generalize. It is mere coincidence that there are less than four ways to split a set of four elements in two equal-sized parts.[4]

Formally, given a program in OBQL + $\mathbf{powerset}_4$, an equivalent program in OBQL + $\mathbf{powerset}_3$ can be obtained by replacing each statement of the form:

$$\mathbf{powerset}_4 \ K[e] \Leftarrow C$$

with the following statements:

[4] In particular, it is readily verified that $1/2\binom{m}{m/2}$, the number of ways to split a set of some even size $m$ in two equal-sized parts, is strictly larger than $m$ if $m > 4$. A similar situation holds for sets of odd size. (E.g., the number of ways to split a set of some odd size $m$ in a part of size $\lceil m/2 \rceil$ and a part of size $\lfloor m/2 \rfloor$ is $\binom{m}{\lfloor m/2 \rfloor}$ wich is at least $m$ if $m > 1$.)

$\mathbf{powerset}_2\ K'[a] \Leftarrow C;$
$K''[b : \{z'_1, z'_2\}] \Leftarrow \bigwedge_{i=1,2} \neg old(z'_i, z'_i) \wedge \neg\exists x : a(z_1, x) \wedge a(z_2, x);$
$K[e' : \{z''_1, z''_2, z''_3\}] \Leftarrow \bigwedge_{i=1,2,3} \neg old(z''_i, z''_i) \wedge$
$\bigwedge_{\substack{i=1,j=2\\ i=2,j=3}} \forall x : (\exists z' : b(z''_i, z') \wedge a(z', x)) \leftrightarrow (\exists z' : b(z''_j, z') \wedge a(z', x));$
$e(z, x) \Leftarrow \exists z'' : \neg old(z'', z'') \wedge \exists z' : e'(z, z'') \wedge b(z'', z') \wedge a(z', x)$
$old(z', z') \Leftarrow z' = z';$
$old(z'', z'') \Leftarrow z'' = z''$

■

We now embark on the proof of Theorem 4.1 for $m \neq 0, 3$. Along the way, we will prove a number of lemmas which we think are interesting in their own right.

As we already mentioned, the theorem is already known for $m = 1$. So, *we will assume for the remainder of this section that* $m \geq 2$. This allows us to make a simplification. Denote by OBQL$^-$ the language of all OBQL programs that do not use object creation statements. Recall that OBQL + $\mathbf{powerset}_m$ is equivalent to OBQL + $m$-bounded set creation. We have:

**Lemma 4.3** OBQL$^-$ + $m$*-bounded set creation is equivalent to* OBQL + $m$*-bounded set creation.*

**Proof.** Given a program in OBQL + $m$-bounded set creation, an equivalent program in OBQL$^-$ + $m$-bounded set creation can be obtained by replacing each object creation statement, of the form

$$K[e_1 : x_1, \ldots, e_k : x_k] \Leftarrow \Phi(x_1, \ldots, x_k)$$

as follows.

If $k = 0$, then $\Phi$ is either true or false. If false, then the statement is simply deleted. If true, it can be replaced with

$$K[e : \{\ \}] \Leftarrow \Phi,$$

for some arbitrary $e$.

If $k = 1$, then the statement can be replaced with

$$K[e_1 : \{x_1\}] \Leftarrow \Phi(x_1).$$

If $k = 2$, then—inspired by Kuratowski's ordered pair construction $[x_1, x_2] \equiv \{\{x_1\}, \{x_1, x_2\}\}$—the statement can be replaced with the following statements. Variables $z_1$, $z_2$ and $z$ are of sort $K_1$, $K_2$ and $K$, respectively.

$$
\begin{aligned}
&K_1[e : \{x_1\}] \Leftarrow \exists x_2 : \Phi(x_1, x_2);\\
&K_2[e : \{x_1, x_2\}] \Leftarrow \Phi(x_1, x_2);\\
&K[e' : \{z_1, z_2\}] \Leftarrow \neg old(z_1, z_1) \wedge \neg old(z_2, z_2) \wedge\\
&\qquad\qquad \exists x_1, x_2 : e(z_1, x_1) \wedge e(z_2, x_1) \wedge e(z_2, x_2) \wedge \Phi(x_1, x_2);\\
&e_1(z, x_1) \Leftarrow \exists z_1 : \neg old(z_1, z_1) \wedge e'(z, z_1) \wedge e(z_1, x_1);\\
&e_2(z, x_2) \Leftarrow \exists z_2 : \neg old(z_2, z_2) \wedge e'(z, z_2) \wedge e(z_2, x_2);\\
&old(z_1, z_1) \Leftarrow z_1 = z_1;\\
&old(z_2, z_2) \Leftarrow z_2 = z_2
\end{aligned}
$$

If $k > 2$, then—inspired by the well-known tuple construction $[x_1, \ldots, x_k] \equiv [x_1, [x_2, \ldots, x_k]]$—the statement can be replaced with the following statements:

$$
\begin{aligned}
&K'[e_2 : x_2, \ldots, e_k : x_k] \Leftarrow \exists x_1 : \Phi(x_1, x_2, \ldots, x_k);\\
&K[e_1 : x_1, e : z'] \Leftarrow \exists x_2, \ldots, x_k : \Phi(x_1, \ldots, x_k) \wedge \neg old(z', z') \wedge \textstyle\bigwedge_{i=2}^{k} e_i(z', x_i);\\
&e_2(z, x_2) \Leftarrow \exists z' : \neg old(z', z') \wedge e(z, z') \wedge e_2(z', x_2);\\
&\vdots\\
&e_k(z, x_k) \Leftarrow \exists z' : \neg old(z', z') \wedge e(z, z') \wedge e_k(z', x_k);\\
&old(z', z') \Leftarrow z' = z'
\end{aligned}
$$

This replacement is repeated until we are reduced to the case $k = 2$. ■

The proof of Theorem 4.1 will be based on the insight that the objects, created by a program in OBQL + $\mathbf{powerset}_m$ when executed on an input instance $I$, can be identified with $m$-bounded hereditarily finite sets with ur-elements in $\mathrm{dom}(I)$, where $\mathrm{dom}(I)$ is the set of all objects occurring in $I$. This identification is formalized next.

**Definition 4.4** *Let $U$ be a set, the elements of which are called* ur-elements. *Then $\mathrm{HF}_m(U)$, the collection of* $m$-bounded hereditarily finite sets with ur-elements in $U$, *is the smallest set satisfying the following two conditions:*

1. $U \subseteq \mathrm{HF}_m(U)$;
2. *for any subset $X$ of $\mathrm{HF}_m(U)$ having cardinality at most $m$, $X \in \mathrm{HF}_m(U)$.*

Let $P$ be a program in OBQL + **powerset**$_m$. Then $P$ is equivalent to a program $Q$ in OBQL$^-$ + $m$-bounded set creation. Let $I$ be an instance to which $Q$ is applied. Let $J$ be the result of this application. We can number the class and property names that are used in $Q$ in some arbitrary but fixed way; let $\#C$ or $\#p$ denote the unique number thus assigned to class name $C$ or property name $p$. We can also number the consecutive statement executions during the application of $Q$ on $I$. Now let $o$ be an object in $J$. We identify $o$ with a member $\tilde{o}$ of $\mathrm{HF}_m(\mathrm{dom}(I))$ in an inductive manner, as follows:

**Definition 4.5**

- *If $o$ is in $I$, then $\tilde{o} := o$.*
- *If $o$ is not in $I$, then $o$ must have been created during the application of $Q$ on $I$, say in the $\ell$th statement execution. Let the statement be of the form:*

  $$K[e : \{x_1, \ldots, x_k\}] \Leftarrow \Phi(x_1, \ldots, x_k)$$

  *Then $o$ is created in function of a set of objects $\{o_1, \ldots, o_k\}$ in $J$, and we define*

  $$\tilde{o} := [\ell, [\#K, [\#e, \{\tilde{o_1}, \ldots, \tilde{o_k}\}]]],$$

*with the understanding that natural numbers are encoded as sets in the following straightforward way: $0 \equiv \emptyset$; $n + 1 \equiv \{n\}$, and that ordered pairs are encoded as sets in the usual Kuratowski way: $[a, b] \equiv \{\{a\}, \{a, b\}\}$.*

In the sequel, we will no longer make a formal distinction between $o$ and $\tilde{o}$.

If $I$ and $J$ are as above, then by the identification of objects in $J$ and elements of $\mathrm{HF}_m(\mathrm{dom}(I))$, every permutation $f$ of the objects of $I$ can be canonically extended to a permutation of the objects of $J$. We can then observe that OBQL + **powerset**$_m$ is *BP-bounded* in the sense of [6]:

**Lemma 4.6** *If $f$ is an automorphism of $I$, i.e., $f$ is a permutation of* $\mathrm{dom}(I)$ *preserving labels and edges, then $f$ is also an automorphism of $J$.*

**Proof.** It suffices to prove the lemma for the simple case where $J$ is obtained from $I$ by application of a single edge addition or $m$-bounded set addition.

Indeed, the lemma will then follow in general by repeated application of this simple case. First, assume $J$ is obtained from $I$ by edge addition:

$$e(x_1, x_2) \Leftarrow \Phi(x_1, x_2).$$

It is well-known that the relational calculus is BP-bounded. Therefore, if $f$ is an automorphism of $I$, then $f$ is also an automorphism of the binary relation over $\mathrm{dom}(I)$ defined by $\Phi$ on $I$, and hence preserves the edges added by the edge addition. Next, assume $J$ is obtained from $I$ by $m$-bounded set creation:

$$K[e : \{x_1, \ldots, x_k\}] \Leftarrow \Phi(x_1, \ldots, x_k) \qquad (k \leq m)$$

Any automorphism $f$ of $I$ is also an automorphism of the $k$-ary relation over $\mathrm{dom}(I)$ defined by $\Phi$ on $I$. Hence, if an object $o$ is created in function of the set $\{o_1, \ldots, o_k\}$, then $\Phi(f(o_1), \ldots, f(o_k))$ will be true in $I$. Therefore, also the object $f(o)$ will be created, in function of the set $\{f(o_1), \ldots, f(o_k)\}$. This completes the proof. ■

We are now ready to present two key lemmas, from which Theorem 4.1 will follow immediately. The lemmas are based upon the following auxiliary notion:

**Definition 4.7** *The* base *of a hereditarily finite set $o$, denoted by $B(o)$, is the set of ur-elements appearing in $o$.*

**Lemma 4.8** *If, for some arbitrary but fixed $m' \geq 2$,* $\mathbf{powerset}_{m'}$ *is expressible in* OBQL+$\mathbf{powerset}_m$ *then there exists an $m$-bounded hereditarily finite set, with base $\{1, \ldots, m'\}$, which is fixed by every permutation of its base.*

**Proof.** Let $S$ be the scheme consisting of one single class name $C$ and no edges. For any natural number $n$, let $I_n$ be the instance over $S$ with $\mathrm{dom}(I_n) = \{1, \ldots, n\}$. So, $I_n$ is a discrete graph consisting of $n$ isolated nodes. As a consequence, every permutation of $\{1, \ldots, n\}$ is an automorphism of $I_n$.

Assume $\mathbf{powerset}_{m'}$ is expressible in OBQL$^-$ + $m$-bounded set creation. Then there exists a program $Q$ in the language which, when applied to $I_n$, is equivalent to the application of the $m'$-restricted powerset operation

$$\mathbf{powerset}_{m'}\ K[e] \Leftarrow C$$

to $I_n$. Let $J_n$ be the result of applying $Q$ to $I_n$. So, there is a one-to-one correspondence between the $m'$-subsets $Z$ of $\{1,\ldots,n\}$ and the $K$-labeled objects $o_Z$ of $J_n$, such that $o_Z$ is linked precisely to the elements of $Z$ by $e$-edges. Furthermore, every object in $J_n$ can be identified with a member of $\mathrm{HF}_m(\{1,\ldots,n\})$, and by Lemma 4.6, every permutation of $\{1,\ldots,n\}$ is an automorphism of $J_n$.

Let us focus on $Z = \{1,\ldots,m'\}$; to keep notation simple, we will write $o_Z$ simply as $o$. Let $f$ be an arbitrary permutation of $\{1,\ldots,n\}$ such that $f(Z) = Z$. Since $f$ is an automorphism of $J_n$, we have an edge $(f(o), e, f(i))$ iff we have an edge $(o, e, i)$, i.e., iff $i \in Z$. Consequently, since $f(Z) = Z$, the set represented by $f(o)$ through its $e$-edges is $Z$. So, necessarily, $f(o) = o$ by the definition of $J_n$. We have thus observed that:

*Every permutation $f$ such that $f(Z) = Z$ fixes $o$.*

Analogously, we can make the observation that also conversely:

*Every permutation $f$ such that $f(o) = o$ satisfies $f(Z) = Z$.*

As a result, if it were the case that $B(o) = Z$, then $o$ itself is the desired $m$-bounded hereditarily finite set.

We can therefore concentrate on the possibility that $B(o) \neq Z$. We distinguish the following cases:

1. $Z$ is a strict subset of $B(o)$. We consider two possibilities:

   (a) $B(o) \neq \{1,\ldots,n\}$. Then we can choose $i \in B(o) - Z$ and $j \in \{1,\ldots,n\} - B(o)$. Clearly, $j \notin Z$. But now consider the transposition $f = (i\ j)$. Clearly, $f(o) \neq o$. However, since $f(Z) = Z$, $f(o)$ must equal $o$, a which is in contradiction with our earlier observations. So this case cannot occur.

   (b) $B(o) = \{1,\ldots,n\}$. Replace each occurrence of an element of $Z$ in $o$ by $\emptyset$; denote the resulting hereditarily finite set by $o'$. Then $o'$ is still $m$-bounded, and $B(o') = \{m'+1,\ldots,n\}$. Since each permutation which is the identity on $Z$ fixes $o$, each permutation of $\{m'+1,\ldots,n\}$ fixes $o'$. Hence, if we put $n = 2m'$ and define the bijection $\psi$ from $\{m'+1,\ldots,2m'\}$ to $\{1,\ldots,m'\}$ by $\psi(i) := i - m'$, we obtain $\psi(o)$ as the desired $m$-bounded hereditarily finite set with base $Z$ which is fixed by each permutation of $Z$.

2. $Z$ is not a subset of $B(o)$, and $Z \cap B(o) \neq \emptyset$. Then we can choose $i \in Z - B(o)$ and $j \in Z \cap B(o)$. As in case (1a) we now arrive at a contradiction. So, this case cannot occur.

3. $Z \cap B(o) = \emptyset$. Again we consider two possibilities:

   (a) $Z \cup B(o) \neq \{1, \ldots, n\}$. Then we can choose $i \in \{1, \ldots, n\} - (Z \cup B(o))$ and $j \in Z$. The transposition $f = (i\ j)$ clearly satisfies $f(o) = o$ and $f(Z) \neq Z$, which is in contradiction with our earlier observations. So this case cannot occur.

   (b) So, necessarily, $Z \cup B(o) = \{1, \ldots, n\}$. Then $B(o) = \{m' + 1, \ldots, n\}$. Each permutation which leaves $Z$ invariant also leaves $o$ invariant. So, in particular, each permutation of $\{m'+1, \ldots, n\}$ leaves $o$ invariant. Hence, with $n = 2m'$ and $\psi$ as in case (1b), we obtain $\psi(o)$ as the desired hereditarily finite set. ■

Theorem 4.1 (for $m \neq 3$) will now follow if we can prove that no $m$-bounded hereditarily finite set exists having base $\{1, \ldots, m+1\}$ which is fixed by each permutation of its base. In fact, we will prove a stronger statement in Lemma 4.9. To this end, we will need to recall some basic facts of group theory.

For a set $X$, denote the group of all permutations of $X$ by $S_X$, and denote the group of all even permutations of $X$ by $A_X$. If $|X| = r$, then $S_X$ and $A_X$ are also written as $S_r$ and $A_r$. If $r \geq 2$ then $|A_r| = r!/2$. Note that any permutation of a set $X$ works also as a permutation of the hereditarily finite sets with ur-elements in $X$ in the canonical manner.

For any group $G$ working on a set $X$, the set $\{g(x) \mid g \in G\}$ is denoted by $G(x)$, and the subgroup $\{g \in G \mid g(x) = x\}$ of $G$ is denoted by $G_x$. By Lagrange's theorem, $|G(x)| = |G|/|G_x|$.

Let $\phi : G \to H$ be a group homomorphism. Then the kernel $\mathrm{Ker}(\phi) = \{g \in G \mid \phi(g) = \mathrm{id}_H\}$ is a normal subgroup of $G$. Furthermore, $\phi$ is injective iff $\mathrm{Ker}(\phi) = \{\mathrm{id}_G\}$. For $r \neq 4$, $A_r$ is simple, i.e., has no nontrivial normal subgroups.

We are now ready for:

**Lemma 4.9** *If $m \neq 4$, then there is no $m-1$-bounded hereditarily finite set whose base is $\{1, \ldots, m\}$ and which is fixed by $A_m$.*

**Proof.** Define the *depth* of a hereditarily finite set as its depth when viewed as a tree in the obvious way. We will prove by induction on $n$ that for each $n$, there is no $m$-bounded hereditarily finite set of depth $n$ with base $\{1, \ldots, m\}$ which is fixed by $A_m$. The basis of the induction, $n = 0$, is trivial. Now let $o = \{o_1, \ldots, o_k\}$, $k < m$, be an $m-1$-bounded hereditarily finite set of depth $n > 0$ with base $Z = \{1, \ldots, m\}$. For the sake of contradiction, assume that $o$ is fixed by $A_m$.

Note that for each $o_i \in o$, $B(o_i) \subseteq B(o) = Z$. First, we show that actually, $B(o_i) = Z$. For, suppose $|B(o_i)| = \ell < m$. Without loss of generality we may assume $B(o_i) \neq \emptyset$ and thus $\ell > 0$. Since for every $f \in A_m$, $f(o_i) \in o$, we have $|o| \geq |A_m(o_i)|$. Clearly, $|A_m(o_i)| \geq |A_m(B(o_i))|$. The latter equals $|A_m|/|(A_m)_{B(o_i)}|$. Every member of $(A_m)_{B(o_i)}$ can be written as a product $f_1 f_2$, where $f_1$ is in $S_{B(o_i)}$ and $f_2$ is in $S_{Z-B(o_i)}$, such that either both factors are even, or both factors are odd. So, $|(A_m)_{B(o_i)}| = \ell!(m-\ell)!/2$, by straightforward calculations. Hence, $k = |o| \geq \binom{m}{\ell}$, which is at least $m$ by our assumptions on $\ell$. But, from the outset, $k < m$; contradiction. Hence, $B(o_i) = Z$.

Since $A_m$ fixes $o$, each $f \in A_m$ induces a permutation on the elements of $o$. Since $o$ has $k$ elements, this yields a natural homomorphism $\phi : A_m \to S_k$. $\mathrm{Ker}(\phi)$ consists of those $f \in A_m$ such that $f(o_i) = o_i$ for each $o_i \in o$. Since $\mathrm{Ker}(\phi)$ is a normal subgroup of $A_m$, the former must be trivial, by the simplicity of the latter. So, there are two possibilities:

- $\mathrm{Ker}(\phi) = \{\mathrm{id}\}$. But then $\phi$ is injective, which would imply that $|A_m| \leq |S_k|$, which is in contradiction with $k < m$.
- $\mathrm{Ker}(\phi) = A_m$. But then each $o_i \in o$, of which we know that $B(o_i) = Z$, would be fixed by $A_m$, which is in contradiction with the induction hypothesis.

Hence, $o$ cannot be fixed by $A_m$, as had to be shown. ■

Note how the above proof relies on the simplicity of $A_r$ for $r \neq 4$. $A_4$ is not simple: a non-trivial normal subgroup of $A_4$ is the group generated by the three permutations (1 2)(3 4), (1 3)(2 4), and (1 4)(2 3). Interestingly, these generators have precisely the same structure as the simulation of 4-sets by 3-sets exhibited in the proof of Proposition 4.2.

# 5 Discussion

Let us conclude by mentioning an interesting open problem. Suppose the known cardinality bound on the set values appearing in an application is $m$. Without supplying the full functionality of $m$-bounded abstraction (or the equivalent $m$-bounded set creation) in the query language, our results indicate that duplicate-free representations will not be achievable in general, i.e., the generation of duplicates will sometimes be unavoidable.

However, one might allow a limited number of duplicates, at the gain of not having to implement abstraction. For example, not having abstraction, one can represent $m$-sets by $m$-ary tuples (which can be created using OBQL's standard object creation), achieving a duplication factor of at most $m!$, which is a limit independent of the particular database instance. This is actually an optimal representation in general, since it can be shown that in the absence of abstraction, in the worst case, as many as $m!$ duplicates per $m$-set will be unavoidable. (This follows from the proof in [19] that $\mathbf{powerset}_m$ is not expressible in OBQL.)

More subtly, one might provide in the query language, not the complete $m$-bounded set creation operation, but a more efficient $k$-bounded one, where $k < m$. For example, if $m = 3$ and $k = 2$, one can represent a 3-set $\{1, 2, 3\}$ as a pair $[1, \{2, 3\}]$. In the worst case, this will yield two other duplicate representations $[2, \{1, 3\}]$ and $[3, \{1, 2\}]$. Hence, the duplication factor is now reduced to 3. Again, it can be shown that this representation is optimal: in the language OBQL + $\mathbf{abstr}|_{m-1}$, $m$ duplicates per $m$-set are unavoidable in general.

This suggests a trade-off between the processing time needed to eliminate duplicates (as provided by abstraction), and the maximum number of duplicates that one can "live with." It would be interesting to develop a cost model to study this trade-off in more detail. (A cost model for duplicate tuple values was described in [4].) Doing so will in particular require the solution of the following problem: in the language OBQL + $abstr|_k$, with $k < m$, how many duplicates per $m$-set are unavoidable in the worst case? In the preceding paragraphs, we answered this question for $k = 1$ and $k = m - 1$, but the general solution remains open.

**Acknowledgment** The first author wishes to thank Michel Van den Bergh for some inspiring discussions in the initial stages of this investigation. We are indebted to Darrell Haile for pointing out the use of the natural group homomorphism $\phi$ in the proof of Lemma 4.9. Marc Gyssens, Jan Paredaens and Inge Thyssens gave useful comments on a previous version of this paper. We also thank an anonymous referee for correcting a few errors in the submitted manuscript.

# References

[1] S. Abiteboul, P. Fischer, and H.-J. Schek, editors. *Nested Relations and Complex Objects in Databases*, volume 361 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[2] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In Clifford et al. [7], pages 159–173.

[3] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences*, 41(2):181–229, 1990.

[4] D. Bitton and D. DeWitt. Duplicate record elimination in large data files. *ACM Transactions on Database Systems*, 8(2):255–265, 1983.

[5] C. Beeri. A formal approach to object-oriented databases. *Data & Knowledge Engineering*, 5(4):353–382, 1990.

[6] A. Chandra and D. Harel. Computable queries for relational database systems. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.

[7] J. Clifford, B. Lindsay, and D. Maier, editors. *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*, volume 18:2 of *SIGMOD Record*. ACM Press, 1989.

[8] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model. In *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, pages 417–424. ACM Press, 1990.

[9] S. Grumbach and V. Vianu. Playing games with objects. In S. Abiteboul and P.C. Kanellakis, editors, *ICDT'90*, volume 470 of *Lecture Notes in Computer Science*, pages 25–38. Springer-Verlag, 1990.

[10] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[11] R. Hull and J. Su. On accessing object-oriented databases: Expressive power, complexity, and restrictions. In Clifford et al. [7], pages 147–158.

[12] R. Hull and J. Su. On the expressive power of database queries with intermediate types. *Journal of Computer and System Sciences*, 43(1):219–237, 1991.

[13] R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proceedings of the 16th International Conference on Very Large Data Bases*. Morgan Kaufmann, 1990.

[14] W. Kim and F.H. Lochovsky, editors. *Object-Oriented Concepts, Databases, and Applications*. Frontier Series. ACM Press, Addison-Wesley, 1989.

[15] G. Kuper and M. Vardi. A new approach to database logic. In *Proceedings of the Third ACM Symposium on Principles of Database Systems*, pages 86–96. ACM Press, 1984.

[16] M. Kifer and J. Wu. A logic for object-oriented logic programming (Maier's O-logic revisited). In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, pages 379–393. ACM Press, 1989.

[17] C. Lécluse, P. Richard, and F. Velez. $O_2$, an object-oriented data model. In H. Boral and P.A. Larson, editors, *1988 Proceedings SIGMOD International Conference on Management of Data*, pages 424–433. ACM Press, 1988.

[18] D. Shipman. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*, 16(10):140–173, 1981.

[19] J. Van den Bussche and J. Paredaens. The expressive power of complex values in object-based data models. Submitted manuscript, 1993, full version of [20].

[20] J. Van den Bussche and J. Paredaens. The expressive power of structured values in pure OODB's. In *Proceedings of the Tenth ACM Symposium on Principles of Database Systems*, pages 291–299. ACM Press, 1991.

[21] J. Van den Bussche and D. Van Gucht. A hierarchy of faithful set creation in pure OODBs. In J. Biskup and R. Hull, editors, *Database Theory—ICDT'92*, volume 646 of *Lecture Notes in Computer Science*, pages 326–340. Springer-Verlag, 1992.