Evaluation and optimization of complex object selections

Jan Van den Bussche
University of Antwerp (UIA), Dept. Math. & Comp. Science
Universiteitsplein 1, B-2610 Antwerp, Belgium
E-mail: vdbuss@ccu.uia.ac.be

Abstract: We provide a general framework for declarative selection operations for complex object databases, based on the safe calculus for complex objects. Within this framework, we consider a class of "single pass-evaluable" selection operations. We show how such selection operations can be succinctly expressed by programs that use only very simple positive existential selections. Also, a syntactic criterion is developed for the *commutation* of two such positive existential selections. These two results are then jointly applied to the problem of *optimizing* complex object selections, which is much more complicated than in classical relational databases.

1 Introduction

Relational database systems enjoy the property of high-level, declarative, *ad-hoc* query languages, with the relational *calculus* as logical basis and the relational *algebra* as operational equivalent. The link between algebra and calculus is most clearly visible in the *selection* operation. In its most general appearance, a selection may be defined as an operation that derives from a set of objects those that satisfy a given logic formula, thus acting as a filter. Selection operations are of fundamental importance in all high-level database management tasks.

Inspired by this, an important issue in the research on *next generation* and *object-oriented* databases is the development of formal, logic-based data models [AK89, Bee90, KLW95, LR89, SS90] which allow the design of high-level query languages (a good example is [BCD89], which is based on [LR89]). Recent efforts in this area typically provide "dynamic" features (like object identity, inheritance, and methods) on top of a core *complex object* data model.

Complex objects are built from atomic ones by arbitrary application of tuple and set constructors, thus extending the relational model of relations containing tuples of atomic values. Ultimately, the whole database structure can be seen as one (very) complex object. A particularly elegant "relational" extension of the relational model that supports complex objects is the *nested relational database model*, where tuple components need not be atomic but may be relations in turn [TF86]. So, tuple and set constructors alternate. Nested relations are in a strong sense "information-wise" equivalent "normal forms" of the seemingly more general unrestricted (i.e., not necessarily alternating) complex objects [HY84]. In Section 2, we review the necessary preliminaries on nested relations.

In Section 3, we introduce a general framework for the study of selection operations for nested relational databases. We concentrate on *generic* selections: this means that values are treated essentially uninterpreted [AU79, Hul86]. Classical relational selections that are generic use only comparisons between values based on (in)equality, like $select[A \neq B]$.

In nested relations, where tuple components can be sets, much more generic comparisons are feasible, like $X \not\subseteq Y$, $A \in X$, $X \cap Y = \emptyset$, $\ldots$. Our framework expresses a large class of generic selection conditions through the calculus for nested relations [AB88, HS88, KV93, PVG88, RKS88]. E.g., $X \not\subseteq Y$ is expressed as: $(\exists A \in X)(\forall B \in Y) A \neq B$. The full power of the nested relational calculus would yield selections of hyperexponential complexity [HS88]. However, we employ only a "safe" fragment of the calculus: all selection operations we consider are computable in polynomial time.

Then we turn to the problem of *evaluation* and *optimization* of complex object selections. In classical relational databases, query evaluators are tuned to handle series of selections very efficiently. Standard optimization algorithms [Ull89] typically "preprocess" the respective selection conditions, which may contain logical connectives, into conjunctive normal form, whereupon the obtained simple conjuncts are processed in a particular order. The strategy for choosing this order ("shuffling") is based on information about storage structure, such as indexes, or distribution of the data. Obviously, the correctness of such strategies relies on the fact that the order in which several selections are applied is irrelevant for the eventual result. Thus, the fact that *any two selection operations commute*, although taken for granted, is fundamental in query optimization.

The situation of selections for nested databases is far more complicated, the main two reasons being *quantifiers* and *deep-level application*:
– Selection conditions can become very involved due to the presence of quantifiers, such that preprocessing a selection into a sequence of "conjuncts" that are simple enough to be treated as atomic operations is not always possible;
– It is strongly desirable to be able to apply selections to instances that appear as complex values deeply within the nested structure. But then, such selection applications can influence the behaviour of other (selection) operations. Two complex objects selections therefore do *not commute* in general.

We give some initial results on the problem of optimizing complex object selections. For a restricted, yet sufficiently general class of selection operations, we provide satisfactory solutions for handling the two complications described above. Inspired by the the classical optimization algorithms, our method also consists of a preprocessing stage, followed by a shuffling stage:
– In Section 4, we preprocess selection operations by expressing them by succinct programs consisting only of very simple, so-called *positive existential* selections, together with two elementary restructuring operations, used for handling temporary storage of intermediate calculations;
– In Section 5.1, we present a syntactic criterion for the commutation of two such positive existential selection operations. Section 5.2 then shows how combining these two results yields a methodology for optimizing a series of complex object selections.

We point out that our results may also find an application in the area of *rule-based systems* [SIG89a, SIG89b, SIG90] for complex objects. Indeed, the if-part of a rule naturally corresponds to a selection condition, and rule triggers might reside on various levels in the complex structure. Therefore, our results are relevant to the issue of *mutual independence* of two or more rules as well as that of *optimization* and *order of evaluation* of a series of rule firings. Some initial investigations are reported in [VdB91].

Finally, in Section 6 we briefly mention related and future work. Among other things, we compare our framework of complex object selections to various selection operations encountered in the literature: even our restricted class of selections turns out to be sufficiently general to express many operations usually considered there.

2 Preliminaries

In this section we briefly introduce a model for working with nested relations, following [PDBGVG89, Chapter 7] to which we refer for more details.

Basically we assume an enumerable set V of *atomic values* and an infinitely enumerable set U of *atomic attributes.*

In the standard relational model (hereafter called the *flat* model), schemes are sets of atomic attributes, and instances are sets of tuples over these attributes. The main idea of nested relations is that attributes can be schemes in turn. As a consequence, tuple components need not be atomic but can be instances as well. Thus the notion of attribute is most naturally extended as follows:

Definition 2.1 *The set* $\mathcal{U}$ *of* attributes *is the minimal set satisfying:*
– U is contained in $\mathcal{U}$;
– every finite subset of $\mathcal{U}$ in which no atomic attribute occurs more than once is an element of $\mathcal{U}$.

Elements of $\mathcal{U} - U$ are called *complex attributes.* Observe that flat relation schemes, being finite subsets of U, are complex attributes. In general we define:

Definition 2.2 *A* scheme *is a complex attribute.*

A scheme R can be viewed as the root of a tree. The children of R are its elements; atomic elements are leafs, while each complex element, being a scheme, is the root of a subtree in turn. From now on, we will not distinguish between a scheme and its associated tree, and use tree terminology when talking about schemes. Thus we define:

Definition 2.3 *For a scheme R, we denote the set of nodes in R by $att(R)$; we extend this to atomic attributes A by putting $att(A) := \{A\}$.*

We stress that hence for any attribute Y, $Y \in att(Y)$.

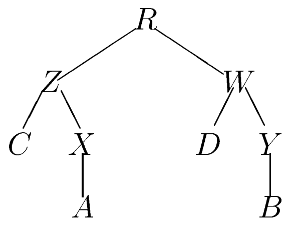
Example 2.4 *Assuming $A, B, C, D \in U$, Figure 1 shows the scheme*

$$R = \{Z = \{C, X = \{A\}\}, W = \{D, Y = \{B\}\}\}$$

We have $att(R) = \{R, Z, C, X, A, W, D, Y, B\}$.

Since values and instances are so closely intertwined, we define them jointly, in the following inductive manner:

Definition 2.5 *Let R be a scheme. The sets $\mathcal{V}$ of* values*, $\mathcal{I}$ of* instances*, $inst(R)$ of* instances over *R and $tup(R)$ of* tuples over *R are the minimal sets satisfying:*
– $\mathcal{V} = V \cup \mathcal{I}$;
– $\mathcal{I} = \bigcup_R inst(R)$;
– $inst(R)$ consists of all finite subsets of $tup(R)$;
– $tup(R)$ consists of all mappings $t : R \to \mathcal{V}$, such that $t(A) \in V$ for each $A \in R \cap U$ and $t(Y) \in inst(Y)$ for each $Y \in R - U$.



$$R = \{Z, W\}$$
$$Z = \{C, X\}$$
$$X = \{A\}$$
$$W = \{D, Y\}$$
$$Y = \{B\}$$

Figure 1: A simple example scheme.

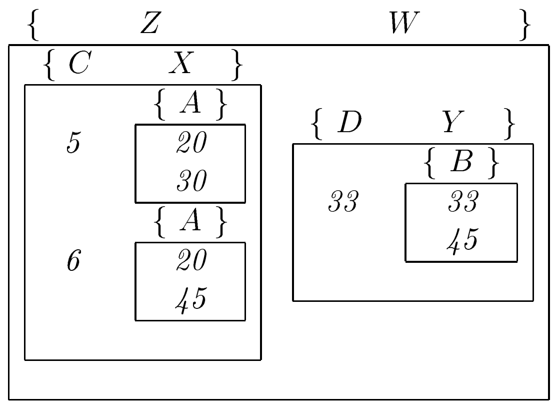


Figure 2: Simple example instance over the scheme of Figure 1.

Example 2.6 *Figure 2 shows an instance over R, containing only one tuple (over R). Concretely one could view a tuple over R as partly describing a robot. The Z-component of a tuple over R is a set of tuples (two in the case of Figure 2), where each such tuple could represent an arm of the robot: the C-component of a tuple over Z might be the length of the arm, while its X-component might be the set of (one-tuples of) possible angles for that arm. Similarly, the W-component of a tuple over R could represent the set of eyes of the robot, where each eye has a focal distance (D) and a set of angles it can look at (Y). The robot of Figure 2 has one eye.*

We will use the scheme of Figure 1 as an abstract running example throughout the paper, discarding the meaning attached to it here.

From the definition, it follows that instances can alternatively be seen as *complex values*. Complex values are typed by the specific scheme over which they are an instance. Even if two instances have different schemes, they can still have the same structure and hence be seen as essentially equal "upon renaming of attributes". In order to formalize this idea, we define:

Definition 2.7
- *A* renaming *is a permutation of U.*
- *Let $X, Y \in \mathcal{U}$. We write $X \cong Y$ if there is a renaming φ such that $Y = X^{\varphi}$.*[1]

For any two atomic attributes A, B, we have $A \cong B$. For complex attributes X, Y, we have $X \cong Y$ iff they are isomorphic when viewed as trees; the renaming φ gives the correspondence between the leafs. Note that there may be several valid choices for φ.

[1] φ is canonically extended to $\mathcal{U}$; also, as is the case here, we will write $\varphi(x)$ as x^{φ}.

Given fixed φ, and complex values r, s over X, Y respectively, we now write $r =_\varphi s$ if $s = r^\varphi$.[2] For simplicity, we will implicitly assume in the sequel that φ is understood whenever two isomorphic complex values are compared and write $r = s$ for $r =_\varphi s$.

3 A general framework of complex object selections

In this section we introduce a general framework of complex object selections. Throughout the remainder of this text, R is an arbitrary but fixed scheme.

3.1 Syntax

First we naturally introduce the logic formulas that will serve as the selection conditions. They form a "safe" fragment of the (tuple) calculus for nested relations. For each $X \in att(R) - U$, we assume an infinitely enumerable set of *(tuple) variables over* X. A variable over X, denoted as t^X, stands for tuples over X.

Definition 3.1 Conditions *are well-formed logic formulas built in the usual way with*

- literals*, of the form:*
  - $t^X|_Z = t'^{X'}|_{Z'}$,[3] *with* $Z \subseteq X$*,* $Z' \subseteq X'$ *and* $Z \cong Z'$*, or*
  - $t^X(A) = \emptyset$*, with* $A \in X - U$*;*
- negation *(*$\neg$*);*
- *the* logical connectives *(*$\vee$*,* $\wedge$*,* $\Rightarrow$*,* $\Leftrightarrow$*);*
- quantifications*, of the form:* $(Qt^X \in t^Y(X))$*, with* $X \in Y$*, and* $Q \in \{\forall, \exists\}$*.*

We will use the following abbreviations for literals:
– $t^X = t'^{X'}$ denotes $t^X|_X = t'^{X'}|_{X'}$,
– $t^X(A) = t'^{X'}(A')$ denotes $t^X|_{\{A\}} = t'^{X'}|_{\{A'\}}$.

Notice that, due to the "safe" format of quantifications, conditions may not contain a literal. The simplest examples for this are the two short conditions: $\neg(\exists t^X \in t^Y(X))$ and $(\exists t^X \in t^Y(X))$. In fact, they can be equivalently written *with* literals, as: $t^Y(X) = \emptyset$ and $\neg t^Y(X) = \emptyset$, respectively. This is the reason why we included this latter kind of literal in our language: it allows us to assume without loss of generality that *every condition contains a literal*, which will turn out to be convenient in our further technical development.

We also point out that the only "constant" we allow in comparisons is the empty set. The results reported here are largely independent of this. For our purposes, it is sufficient to treat '$t(A) = c$' simply as '$t(A) = t(B)$', where B is an extra tuple component with constant value c.

[2] Any renaming can be naturally extended to $\mathcal{V}$ by making it the identity on V.
[3] If f is a mapping on a set S, and $S' \subseteq S$, then we denote the restriction of f to S' by $f|_{S'}$.

3.2 Semantics

Whether a condition makes sense as a condition for complex object selections depends on where in the scheme tree the selection takes place. Let $W \in att(R) - U$ and r an instance over R. If $W \neq R$, then a tuple t over W can appear several times in r. Given such an appearance α_t of t, certain tuples appearing in r correspond to α_t in a unique way. First of all this is t itself. Furthermore, if Z is the parent of W, there is a unique tuple $\bar{t}$ over Z, in which α_t appears. More generally, this works for any ancestor Z of W in R (i.e., $W \in att(Z)$).

Thus, an *appearance* α_t of a tuple t over W in r corresponds to a mapping

$$\alpha_t : \{Z \mid W \in att(Z)\} \to \bigcup_{Z \mid W \in att(Z)} tup(Z)$$

such that: $\alpha_t(W) = t$; $\alpha_t(Z_1) \in \alpha_t(Z_2)(Z_1)$ if $Z_1 \in Z_2$; and $\alpha_t(R) \in r$.

Example 3.2 *Consider our example instance r (Figure 2). r contains one tuple t_1 over R; $t_1(Z)$ contains two tuples: $t_{11} = [C : 5, X : \{[A : 20], [A : 30]\}]$ and $t_{12} = [C : 6, X : \{[A : 20], [A : 45]\}]$. The tuple $t = [A : 20]$ over X is an element of $t_{11}(X)$ as well as of $t_{12}(X)$. So t appears twice in r. Let α_t be the appearance of t in t_{11}. So, $\alpha_t(Z) = t_{11}$, and $\alpha_t(R) = t_1$. In contrast, there is no tuple over Y that naturally corresponds to α_t. This is because $X \notin att(Y)$: $\alpha_t(Y)$ is not defined.*

Definition 3.3 *Let $W \in att(R) - U$. A* selection condition over *W is a condition P such that for each free variable occurrence t^Z, $W \in att(Z)$.*

So, one condition can be a selection condition over several W's.

Definition 3.4 *Let $W \in att(R) - U$, r be an instance over R, and α_t be an appearance of a tuple t over W in r. Let P be a selection condition over W. Then $P(\alpha_t)$ is the boolean value obtained by evaluating P over r, when we substitute each free variable occurrence t^Z by $\alpha_t(Z)$.*

By the above discussion, $P(\alpha_t)$ is well-defined. Obviously, every selection condition has an equivalent form where for each X there is at most one free variable over X: indeed, two free variables of the same sort play the same role, and hence can be identified. We finally state:

Definition 3.5 *Let W, r, P be as above. The* selection *$select_W[P](r)$ is the instance over R, obtained from r by filtering out those appearances α_t of tuples t over W in r for which $P(\alpha_t)$ is false.*

Example 3.6 *Let R be our example scheme (Figure 1). Then $P_1 \equiv \neg t^R(Z) = t^R(W)$ is a selection condition over R, as well as*

$$P_2 \equiv (\forall t^Z \in t^R(Z))(\exists t^W \in t^R(W))t^Z = t^W$$

The selection $select_R[P_2]$ can be paraphrased as "$select_R[t^R(Z) \subseteq t^R(W)]$". Similarly, the selection condition over W: $P_3 \equiv (\exists t^Y \in t^W(Y))t^Y(B) = t^W(D)$ can be paraphrased as

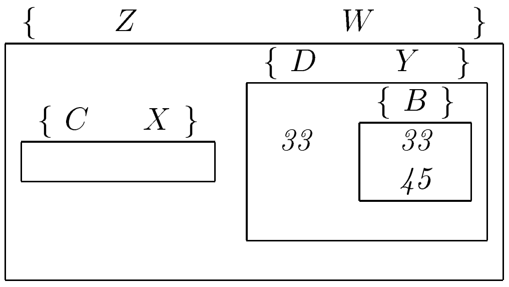


Figure 3: Result of applying $select_Z[P_4]$ to the instance of Figure 1.

"$t^W(D) \in t^W(Y)$". We invite the reader to check that, for the example instance r of Figure 2, $select_W[P_3](r) = r$. A selection condition over Z involving a logical connective is P_4:

$$(\forall t_1^Z \in t^R(Z))\big(\neg t_1^Z = t_2^Z \Rightarrow (\forall t_1^X \in t_1^Z(X))(\forall t_2^X \in t_2^Z(X))\neg t_1^X(A) = t_2^X(A)\big)$$

t^R and t_2^Z are the only free variables of P_4. The selection $select_Z[P_4]$ retains only those appearances of tuples over Z whose X-component is disjoint with that of all other tuples belonging to the Z-value it belongs to. Using the notations of Example 3.2, if we apply this selection to the example instance r, both t_{11} and t_{12} will be deleted: the result is shown in Figure 3.

Selection conditions form a fragment of the tuple calculus for nested relations; in particular, the formulas are all *safe* [RKS88] (called *strictly safe* in [AB88]). This implies that selections can be computed in the strictly safe algebra, i.e., without the powerset operator, for nested relations. It can be shown that the strictly safe algebra only expresses queries computable in polynomial time.

To end this section, please note that our model of selection operations is not meant to serve as a user-level specification language, but rather as a uniform framework for reasoning about complex object selections.

4 Evaluation of a restricted class of selections

In this section, we discuss three restrictions on selection conditions. Conditions satisfying these three conditions are called *restricted.* As a very simple subclass of restricted conditions we introduce positive existential conditions. Finally, we show that selections using arbitrary restricted conditions can be expressed in terms of selections using only positive existential conditions.

Definition 3.1 (of conditions) is very general. In order to be able to obtain specific results, we now restrict the class of conditions as follows. Throughout, we are considering selection conditions over a fixed $W \in att(R) - U$.

1. *No logical connectives allowed.*

Note that this immediately reduces conditions to the general format of a sequence of (possibly negated) quantifications, followed by a literal.

Our second restriction essentially states that the selection operation can be evaluated in a single "scan". So, double quantifications are not allowed, and neither are quantifications over sets containing tuples over W, since the selection operation itself already performs a scan on this level.

2. *For each $X \in att(R) - U$ there must be at most one quantification over X, i.e., of the form $(Qt^X \in t^Y(X))$. Furthermore, if $W \in att(X)$, then no such quantification can occur.*

Our third and final restriction is slightly more technical:

3. *Let $\mathcal{Q} \equiv (Qt^X \in t^Y(X))$ be a quantification that either is universal or immediately precedes an occurrence of negation. Then the condition following $\mathcal{Q}$ must be a selection condition over X (Definition 3.3).*

This restriction has the intuition that quantified negative conditions and universally quantified conditions must be "local" to this quantification.

Definition 4.1 *A condition is called* restricted *if it satisfies the restrictions (1), (2) and (3) above.*

The class of restricted selection operations is still sufficiently powerful to express many selections encountered in the literature; see Section 6.

Example 4.2 *Reconsider Example 3.6. The conditions P_1, P_2, P_3 are restricted; P_4 however is clearly not. As a more subtle example, also in the context of our example scheme of Figure 1, the following selection condition over W satisfies restrictions (1) and (2) but not (3):*

$$(\exists t^Z \in t^R(W))(\forall t^Y \in t^W(Y))(\exists t^X \in t^Z(X))t^Y = t^X$$

Indeed, t^Z occurs free in the condition $(\exists t^X \in t^Z(X))t^Y = t^X$, but $Y \notin att(Z)$.

The motivation for considering restricted selections will be clear from Theorem 4.7 and the remarks following thereafter. Of crucial importance in this respect are the *positive existential conditions*:

Definition 4.3 *A condition is called* positive existential *if it satisfies restrictions (1) and (2) above, has only existential quantifications, and contains no negation.*

Observe that positive existential conditions are restricted, since restriction (3) is voidlessly satisfied.

Although the expressiveness of positive existential conditions seems rather limited, we will see in Theorem 4.7 that any restricted selection operation is expressible by a program consisting only of positive existential selections, together with the two elementary operations *copy* and *singleton*.

These last two operations are trivial restructurings, defined as follows:

Definition 4.4 *Let r be an instance over R, $W \in att(R) - U$, and $Y \subseteq W$.*

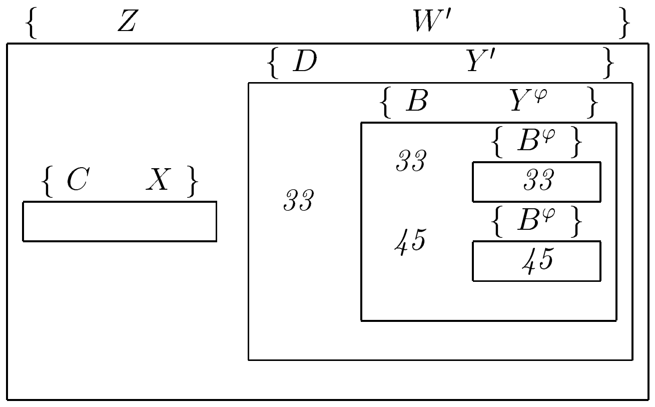


Figure 4: The result of applying $copy[Y \to Y^{\varphi}]; singleton[Y^{\varphi}]$ to the instance of Figure 3.

- *Let φ be a renaming such that $att(Y^{\varphi}) \cap att(R) = \emptyset$. The operation $copy[Y \to Y^{\varphi}](r)$ yields the instance r' over R', where R' is obtained from R by replacing W by $W' = W \cup Y^{\varphi}$, and r' is obtained from r by replacing each tuple t over W appearing in r by the tuple t' over W' defined by: $t'|_W := t|_W$ and $t'|_{Y^{\varphi}} := t|_Y$.*
- *The operation $singleton[Y](r)$ yields the instance r' over R', where R' is obtained from R by replacing W by $W' = (W - Y) \cup \{Y\}$, and r' is obtained from r by replacing each tuple t over W appearing in r by the tuple t' over W' defined by: $t'|_{W-Y} := t|_{W-Y}$ and $t'(Y) := \{t|_Y\}$.*

Example 4.5 *See Figure 4, where $W' = \{D, Y'\}$ and $Y' = Y \cup \{Y^{\varphi}\}$.*

In order to be able to remove copies, we need also the ability to project out a set Y of attributes: because of the particular usage we will make of such projections, we will call them *copy-removals* and denote them by $remove[Y]$.

Example 4.6 *Before formally stating Theorem 4.7, we already informally indicate how the copy and singleton operations are used in the simulation of arbitrary restriction selections by positive existential ones.*

Consider again the example scheme R of Figure 1, and consider the selection

$$select_W[(\forall t^Y \in t^W(Y)) \neg t^Y(B) = t^W(D)]$$

The following equivalent program eliminates the universal quantification: (with φ an appropriate renaming)

```
$copy[\{Y\} \to \{Y^{\varphi}\}]$;
  /* denote the scheme $W \cup \{Y^{\varphi}\}$ by $W'$ */
$select_{Y^{\varphi}}[\neg t^{Y^{\varphi}}(B^{\varphi}) = t^{W'}(D)]$;
$select_{W'}[t^{W'}(Y) = t^{W'}(Y^{\varphi})]$;
$remove[\{Y^{\varphi}\}]$
```

That is, we apply the unquantified selection to a copy of each set, after which we simply test whether the copy has remained unchanged, since this is exactly the case if the condition is universally true in the set.

The first selection of the above program still contains an occurrence of negation. This is in turn eliminated if we substitute this first selection in turn by the following equivalent program: (with ψ an appropriate renaming)

```
$copy[Y^{\varphi} \rightarrow Y^{\psi}]$;
$singleton[Y^{\psi}]$;
  /* denote the scheme $Y^{\varphi} \cup \{Y^{\psi}\}$ by $Y'$ */
  /* denote the scheme $\{D, Y, Y'\}$ by $W''$ */
$select_{Y^{\psi}}[t^{Y^{\psi}}(B^{\psi}) = t^{W''}(D)]$;
$select_{Y'}[t^{Y'}(Y^{\psi}) = \emptyset]$;
$remove[\{Y^{\psi}\}]$
```

That is, we apply the unnegated selection operation to a singleton-copy of each tuple, after which we can simply test each result for emptiness, since this is exactly the case if the condition does not hold for the tuple.

Using the techniques illustrated in the above example we can show (proof ommitted):

Theorem 4.7 For each restricted selection operation there is an equivalent program consisting only of positive existential selections, copy, and singleton, up to copy-removal.

The length of this program is proportional to the number of universal quantifiers and negations that occur in the condition.

As the above example suggests, Theorem 4.7 has a constructive proof, which provides us with an effective algorithm for expressing restricted selections by positive existential ones. Thus, an immediate application of the theorem is in the *evaluation of complex object selections*. The following remarks are in order here:

- Positive existential selections, being so simple, have a great chance of finding efficient implementations. In particular, if only atomic values are compared, they can be efficiently evaluated using slight adaptations of known techniques, such as those described in [Bid87, BRS82, PSS+87];
- the operations copy, singleton and copy-removal are merely conceptual representations of the flow of control in the program, and hence can be argued to have neglectable cost;
- the number of operations produced by the algorithm is linearly proportional (with low coefficients) to the number of occurring negations and universal quantifications in the given restricted selection condition.

The last item is of particular interest. Indeed, consider the following (extreme) example.

Example 4.8 *Let P be a selection condition, having the rough format: $\neg\forall\forall\forall\neg\ell$ with ℓ a literal. Then 2 negations and 3 universal quantifiers occur in P. Hence, the evaluation algorithm of Theorem 4.7 produces an equivalent program of about 18 operations. However, if P is first rewritten in the equivalent format: $\exists\exists\exists\ell$ then we immediately obtain a positive existential condition, i.e., a program of 1 operation. As a more subtle example, a (sub)condition of the format $\neg\exists\neg\cdots$ can be rewritten as $\forall\cdots$, eliminating one step in the evaluation algorithm.*

Thus, the problem comes up of rewriting the selection condition into an equivalent (restricted) one whose $\forall$-$\neg$-*degree*, i.e., the total number of occurring negations and universal quantifiers, is minimal. This problem can be naturally restated as follows. With a restricted condition one can associate a string over the alphabet $\{\forall, \exists, \neg\}$, by looking only at the quantifier of each quantification and ignoring the literal. The problem is to find an *equivalent* string of minimal degree. Here, two strings are equivalent if they can be transformed to each other using the well-known De Morgan equations: $\forall\neg = \neg\exists$, $\exists\neg = \neg\forall$ and the rule $\neg\neg = \lambda$ (the empty string). Note that the two De Morgan rules can be replaced by the single equation $\neg\exists\neg = \forall$.

The following procedure *minimize*(P), with P a restricted selection condition, solves the problem. We use the following terminology: bringing a string in *normal form* w.r.t. a certain rewrite rule means repeatedly and exhaustively applying the rule to the string.

1. Bring P in normal form w.r.t. $\forall \to \neg\exists\neg$, and call the result P_1;
2. Bring P_1 in normal form w.r.t. $\neg\neg \to \lambda$, and call the result P_2;
3. Bring P_2 in normal form w.r.t $\neg\exists\neg \to \forall$, *applying the rule from left to right.* The result is *minimize*(P).

Clearly, *minimize* runs in time linear in the length of P. We can show: (proof omited)

Proposition 4.9 *minimize*(P) is of minimal $\forall$-$\neg$-degree.

Of course, one must also check that *minimize*(P) is still restricted. It can be shown that this is indeed the case, provided that P does not contain "unnecessary" quantifications, but we omit the details here.

Example 4.10 *Consider the string $P = \forall\exists\forall$. Then $P_2 = \neg\exists\neg\exists\neg\exists\neg$, and minimize$(P) = P$: P is indeed of minimal degree itself. If, however, in Step 3 we would not start at the left and rewrite P_2 into $\neg\exists\forall\exists\neg$, we obtain a string of non-minimal degree, one higher than the degree of P.*

We end this section with a remark on *conjunctions* of selection conditions. In the condition of a restricted selection, logical connectives, such as conjunction, are disallowed (restriction (1)). We can however slightly relax this restriction: consider a selection operation $select_W[P_1 \ \& \ P_2]$, where P_1, P_2 are both restricted selection conditions over W. Then obviously, this selection can be alternatively written as the composition $select_W[P_1]; select_W[P_2]$ and we can apply Theorem 4.7 to both parts.

In fact, the order of the above composition is irrelevant for the eventual result: we could do the same as well with $select_W[P_2]; select_W[P_1]$. This is clearly due to the fact that both selections apply over the same W. Indeed, in general, two (restricted) selections $select_{W_i}[P_i]$, $i = 1, 2$, do not necessarily commute. In the next section we look more closely into this problem, and its natural connection to the problem of *optimizing* complex object selections.

5 Commutation and optimization of selection operations

In this section, we consider the problem of *commutation* of selection operations. First we give a decision procedure for commutation of two positive existential selections. Then, exploiting Theorem 4.7, we look how this commutation criterion can be used in the problem of optimizing multiple selection operations. Throughout we only consider restricted selection conditions.

5.1 Commutation of positive existential selections

Unlike the case of flat relational databases, in nested databases, the particular order in which a series of selection operations is applied can affect the final result. The reason for this is of course that selections can apply at arbitrary places in the scheme tree, which results in a complex interaction.

Example 5.1 *Consider our example scheme R of Figure 1, and the instance r over R of Figure 2. r contains one tuple, t. $t(W)$ in turn also contains one tuple: $t_1 = [D : 33, Y : \{[B : 33], [B : 45]\}]$.*

Now consider the two selection operations S_1, S_2:

$$S_1 = \mathit{select}_W[(\exists t^Y \in t^W(Y))t^Y(B) = t^W(D)]$$

$$S_2 = \mathit{select}_Y[(\exists t^Z \in t^R(Z))(\exists t^X \in t^Z(X))t^Y(B) = t^X(A)]$$

Let us compute both $S_2(S_1(r))$ and $S_1(S_2(r))$. Clearly, S_1 leaves r unchanged. Applying S_2 to $S_1(r) = r$ results in the removal of $[B : 33]$ from $t_1(Y)$, whence applying S_1 to $S_2(r)$ yields an instance r' containing only one tuple t', with $t'(Z) = t(Z)$ and $t'(W) = \emptyset$. We observe that $S_1(S_2(r)) \neq S_2(S_1(r))$. Hence, S_1 and S_2 do not commute for all possible instances.

Consider on the other hand the selection $S_3 = \mathit{select}_Y[t^Y(B) = t^W(D)]$. Then it can be seen that S_1 and S_3 always commute.

In the above example, we found combinations of selections that did commute and others that did not. As pointed out in the Introduction, having a criterion for commutation to our disposal is of critical importance in the *optimization* of selections. It indeed turns out that it is possible to decide, given two positive existential selection operations, whether or not they commute in general.

In order to be able to state our commutation criterion in an elegant way, we simplify the format of literals a little. Literals of the form $t^X(A) = \emptyset$ are treated as if of the form $t^X(A) = t^X(B)$, where $B \cong A$ is an extra attribute with constant value $\emptyset$. Further, literals of the form $t^X|_Z = t'^Y|_W$, where Z, W are not singletons, are reduced to literals involving only single attributes by applying the restructuring:

$\mathit{copy}[Z \to Z^\varphi]$; $\mathit{copy}[W \to W^\varphi]$;
$\mathit{singleton}[Z^\varphi]$; $\mathit{singleton}[W^\varphi]$;

So, the subtuples over Z and W are copied in a singleton complex value over Z^φ and W^φ, respectively, and we can now use the simple literal $t(Z^\varphi) = t'(W^\varphi)$.

Now let $S_i = select_{W_i}[P_i]$, $i = 1, 2$, be two positive existential selections. As a result of the above discussion, we may assume without loss of generality that the literal of P_i is of the form $t^{X_i}(A_i) = t'^{Y_i}(B_i)$. To exclude singular cases, we furthermore assume that $A_i \neq B_i$.

Definition 5.2 *S_i is said to* influence *S_j if:*

$$(W_i \leftrightarrow A_j \mathit{\ or\ } W_i \leftrightarrow B_j) \mathit{\ and\ } W_j \notin att(W_i)$$

where, for two attributes $Z, Z' \in att(R)$, we write $Z \leftrightarrow Z'$ if Z and Z' lie on a common path in R.

Intuitively, S_i influences S_j if the set of A_j- and B_j-values from perspective W_j may be altered due to deletions at perspective W_i. The extra condition that $W_j \notin att(W_i)$ generalizes the situation in flat relational databases (where necessarily $W_i = W_j = R$); if $W_j \in att(W_i)$, deletions at level W_i also delete tuples at level W_j, so in that case S_i does not influence S_j, but "cooperates" with it (though in a rather crude manner).

More precisely, we have established (proof ommitted):

Theorem 5.3 If $\{A_1, B_1\} = \{A_2, B_2\}$, then S_1 and S_2 always commute. Otherwise, S_1 and S_2 always commute if and only if they do not influence each other.

Corollary 5.4 Given two positive existential selection conditions, it is decidable whether or not they always commute, in time polynomial in the size of the scheme R.

Example 5.5 *Reconsider the selections S_1, S_2, S_3 of Example 5.1. We can now use Theorem 5.3 to verify commutation:*

- *S_1 and S_2 do not commute, since S_2 influences S_1. Indeed, $W_1(= W) \notin att(W_2(= Y))$ and $Y \leftrightarrow A_1(= B)$.*
- *S_1 and S_3 commute, since $\{A_1(= B), B_1(= D)\} = \{A_3(= D), B_3(= B)\}$.*
- *Finally, S_2 and S_3 also commute, since $W_2 = W_3 = Y$, and $Y \in att(Y)$; hence, it is impossible for S_2 and S_3 to influence each other. Actually,* any *two selections S_1, S_2 for which $W_1 = W_2$ commute, for the same reason.*

5.2 Towards optimizing complex object selections

We now enlarge our focus from the problem of commutation of positive existential selections to the more general problem of commutation and optimization of restricted selection operations.

A first observation is that Theorem 5.3 does not straightforwardly extend to the general case of restricted selection operations, as illustrated next:

Example 5.6 *In Example 5.5 we used Theorem 5.3 to deduce that the selections S_1, S_3 of Example 5.1 commute, since $\{A_1, B_1\} = \{A_3, B_3\}$. Consider however the negated version of S_3: $S_4 = select_Y[\neg t^Y(B) = t^W(D)]$. Clearly S_4 is not positive existential, and although $\{A_1, B_1\} = \{A_4, B_4\}$, S_1 and S_4 do* not *always commute, as is readily seen.*

Actually, a criterion for commutation of general selection operations would be of lesser interest to optimization. Indeed, as the operations become more complex, simple "global" strategies of choosing an execution order might be too weak. Instead, we will propose a strategy in which we descend into the complex structure of the selections, down to the level of single positive existential ones, and apply Theorem 5.3 there. This will be possible with the aid of Theorem 4.7. Thus, let $S_i = \mathit{select}_{W_i}[P_i]$, $1 \leq i \leq n$, of the form as above, be n restricted selections. The problem is to optimize the program $S_1; \cdots; S_n$. By Theorem 4.7, each S_i can be expressed as a sequence ζ_i of positive existential selections, together with copy, singleton and copy-removal operations.

Our approach is as follows. First, we apply all copies and singletons occurring in the ζ_i, beforehand, and independently, i.e., care is taken that all corresponding renamings do not coincide. We have thus obtained an enlarged scheme R', which equals R "with copies".[4] Each S_i is now expressed as the subsequence ζ_i^-, consisting of all (positive existential) selection operations in ζ_i. In analogy with the problem of optimizing selections in flat relational databases, we state the problem of *optimizing the series of complex object selections* $S_1; \cdots; S_n$ as follows: *Merge the sequences* ζ_i^- *in such a way that the most optimal evaluation sequence is obtained.*

As in standard algorithms for optimizing selections in relational databases [Ull89], strategies for doing this will typically rely on information about storage structure, such as indexes, or distribution of the data. However, a considerable complication, in contrast with the situation in flat relational databases, is that here we have to deal with the issue of commutation: an optimization is correct only if the obtained merging always yields the same result as the original program $\zeta_1^-; \cdots; \zeta_n^-$. Here, Theorem 5.3 comes in. The crucial observation is: every merging can be obtained by starting from the concatenation of the original sequences, $\zeta_1^-; \cdots; \zeta_n^-$, and then "pushing" the positive existential selection operations of ζ_i^- through those of ζ_j^-, $j < i$. At each push, we can effectively check whether or not the two selections may be effectively switched without altering the final result of the program. By first minimizing the length of the ζ_k's, as explained in Section 4, we can significantly reduce the search space, i.e., the number of possible mergings to consider.

Example 5.7 *As a simple example, recall the selections* S_1, S_4 *of Examples 5.1 and 5.6. We have for* ζ_4^-*:*

S_{41}: $\mathit{select}_{Y^\varphi}[t^{Y^\varphi}(B^\varphi) = t^{W'}(D)]$;
S_{42}: $\mathit{select}_{Y'}[t^{Y'}(Y^\varphi) = t^{Y'}(Y^\psi)]$;

Here, Y', W' *correspond to* Y, W *in the scheme* R' *with copies: (*φ, ψ *are appropriate renamings)*

$$R' = \{Z, W' = \{D, Y' = \{B, Y^\varphi, Y^\psi\}\}\}$$

where Y^ψ *serves as "tag" for the empty set (so* S_{42} *really expresses* $t^{Y'}(Y^\varphi) = \emptyset$*, as explained in Section 5.1). Since* S_1 *is positive existential,* ζ_1^- *is* S_1 *itself, slightly adapted to work on instances over* R'*:*

S_{11}: $\mathit{select}_{W'}[(\exists t^{Y'} \in t^{W'}(Y'))t^{W'}(D) = t^{Y'}(B)]$

[4] Thus, we delay the copy-removals until the end; in order to make this work, the selection conditions have to be slightly adapted. We do not go into detail about this here; Example 5.7 gives a simple illustration.

Now by Theorem 5.3, S_{11} and S_{42} do not commute; however S_{11} and S_{41} do. Thus, for evaluating $S_4; S_1$ we have no choice but to use the unmerged sequence composition $\zeta_4^-; \zeta_1^-$. However, for evaluating $S_1; S_4$ we can choose between $S_{11}; S_{41}; S_{42}$ and $S_{41}; S_{11}; S_{42}$.

6 Discussion

In the framework of a general calculus-based theory for complex object selection operations, we gave some initial results concerning the evaluation and optimization of a restricted class of "single-pass" operations.

Other rather general systems of complex object selections have been proposed in the literature. The Verso super selection was introduced in [Bid87]. As in our approach, general logic formulas are used there as selection conditions; however, Verso selections work only on the top level. Actually, the focus of [Bid87] is more on succinctly expressing relational tableau queries through Verso super selection; this is possible since Verso databases correspond to universal relation databases.

In [AB88, BCD89, BK90, SS86] (among others), a more algebraic approach is taken in defining powerful selection operations, by allowing other algebra operations to appear in selection conditions, as in "$\mathit{select}[\pi[C](Z) \cap \pi[D](W) = \emptyset]$". The optimization problem may then be attacked by finding algebraic equivalences, stating how the various operations commute and interact. Results in this direction can be found in [Col89, Sch86]. In comparison, our approach focuses more specifically on the particular structure of declarative selection operations. Thus, we obtain rather detailed results, which may also be more readily applicable to other logic-based environments, like rule-based systems. Moreover, our model of selections is not tied to a particular query specification language, but is also translatable to algebraic formalisms. So, both the algebraic as our methodology can be used complementary.

And conversely, the restricted class of selections we considered in Sections 4 and 5 is able to capture most selection operations usually encountered in the literature (and many others too). A typical representative of these are the *extended selections* defined in [SS86]. Their corresponding conditions are formulas built from comparisons based on $=$, $\subseteq$, $\in$ (including "dynamic constants"), negation, logical connectives, and a limited form of existential quantification (due to the recursive use of projection and selection within the conditions). It can be shown (cfr. Example 4.2 and the comments at the end of Section 4 and the present section) that all such conditions are restricted in our sense, or can be equivalently expressed by restricted ones.

An obvious direction to extend our results is to extend the class of restricted selection conditions. If one still wants an analogue of Theorem 4.7, it may then be necessary to extend the set of "auxiliary" operations, in our case the copy and singleton operations. A simple example of this idea would be to add disjunction through selections of the form $\mathit{select}_W[P_1 \text{ or } P_2]$, where P_1, P_2 are restricted. One can express this operation by the program:

$\mathit{copy}[\{W\} \to \{W^\varphi\}]; \mathit{copy}[\{W\} \to \{W^\psi\}];$
$\mathit{select}_{W^\varphi}[P_1^\varphi]; \mathit{select}_{W^\psi}[P_2^\psi];$
$\mathit{union}[W^\varphi \cup W^\psi \to W]$

where the *union* construct is an additional auxiliary operation, replacing each W-instance by the union of its corresponding copies.

Acknowledgment

Thanks go to Jan Paredaens, for inspiring discussions, and to Marc Gyssens, Peter Peelman, Inge Thyssens, and some anonymous referees, for their helpful comments.

References

[AB88] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. Technical Report 846, INRIA, 1988.

[AK89] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In Clifford et al. [CLM89], pages 159–173.

[AK90] S. Abiteboul and P.C. Kanellakis, editors. *ICDT'90*, volume 470 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[AU79] A.V. Aho and J.D. Ullman. Universality of data retrieval languages. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 110–120, 1979.

[BCD89] F. Bancilhon, S. Cluet, and C. Delobel. Query languages for object-oriented database systems. In R. Hull, R. Morrison, and D. Stemple, editors, *Proceedings of the Second International Workshop on Database Programming Languages*, pages 122–138. Morgan Kaufmann, 1989.

[Bee90] C. Beeri. A formal approach to object-oriented databases. *Data & Knowledge Engineering*, 5(4):353–382, 1990.

[Bid87] N. Bidoit. The Verso algebra or how to answer queries with fewer joins. *Journal of Computer and System Sciences*, 35(3):321–364, 1987.

[BK90] C. Beeri and Y. Kornatzky. Algebraic optimization of object-oriented query languages. In Abiteboul and Kanellakis [AK90], pages 72–88.

[BRS82] F. Bancilhon, P. Richard, and M. Scholl. On line processing of compacted relations. In *Proceedings 8th International Conference on VLDB*, pages 263–269, 1982.

[CLM89] 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.

[Col89] L.S. Colby. A recursive algebra and query optimization for nested relations. In Clifford et al. [CLM89], pages 273–283.

[HS88] R. Hull and J. Su. On the expressive power of database queries with intermediate types. In PODS [POD88], pages 39–51.

[Hul86] R. Hull. Relative information capacity of simple relational schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986.

[HY84] R. Hull and C.K. Yap. The format model, a theory of database organization. *Journal of the ACM*, 31(3):518–537, 1984.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, July 1995.

[KV93] G. Kuper and M. Vardi. The logical data model. *ACM Transactions on Database Systems*, 18(3):379–413, 1993.

[LR89] C. Lécluse and P. Richard. Modeling complex structures in object-oriented databases. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, pages 360–368. ACM Press, 1989.

[PDBGVG89] J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The Structure of the Relational Database Model*, volume 17 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1989.

[POD88] *Proceedings of the Seventh ACM Symposium on Principles of Database Systems*. ACM Press, 1988.

[PSS$^+$87] H.-B. Paul, H.-J. Schek, M.H. Scholl, G. Weikum, and U. Deppisch. Architecture and implementation of the Darmstadt Database kernel system. In U. Dayal and I. Traiger, editors, *Proceedings of the ACM SIGMOD 1987 Annual Conference*, volume 16:3 of *SIGMOD Record*, pages 196–207. ACM Press, 1987.

[PVG88] J. Paredaens and D. Van Gucht. Possibilities and limitations of using flat operators in nested algebra expressions. In PODS [POD88], pages 29–38.

[RKS88] M. Roth, H. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*, 13(4):389–417, 1988.

[Sch86] M.H. Scholl. Theoretical foundations of algebraic optimization utilizing unnormalized relations. In G. Ausiello and P. Atzeni, editors, *ICDT'86*, volume 243 of *Lecture Notes in Computer Science*, pages 409–420. Springer-Verlag, 1986.

[SIG89a] SIGMOD. Session on triggers and derived data. In Clifford et al. [CLM89].

[SIG89b] SIGMOD. Special issue on rule management and processing in expert database systems. *SIGMOD Record*, 18(3), 1989.

[SIG90] SIGMOD. Session on rule processing systems. In H. Garcia-Molina and H.V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, volume 19:2 of *SIGMOD Record*. ACM Press, 1990.

[SS86] H.-J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[SS90] M.H. Scholl and H.-J. Schek. A relational object model. In Abiteboul and Kanellakis [AK90], pages 89–105.

[TF86] S. Thomas and P. Fischer. Nested relational structures. In P. Kanellakis, editor, *The Theory of Databases*, pages 269–307. JAI Press, 1986.

[Ull89] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, 1989.

[VdB91] J. Van den Bussche. Active complex object databases. In J. Göers, A. Heuer, and G. Saake, editors, *3rd Workshop on Foundations of Models and Languages for Data and Objects*, number 91/3 in Informatik-Bericht, pages 103–113. TU Clausthal, 1991. To appear.