Algebraic Unnesting for Nested Queries

Quan Wang[1] David Maier[1] Leonard Shapiro[2]
[1]Oregon Graduate Institute of Science and Technology
[2]Portland State University
[1]{quan, maier}@cse.ogi.edu [2]len@cs.pdx.edu
November 8, 1999

ABSTRACT Both relational and object query languages allow nested queries (queries with sub-queries). Nested queries are typically inefficient to execute in their original form [K82], and are difficult to optimize. Most existing query unnesting techniques are based on source-to-source, query graph, or calculus transformations; few are based on algebraic transformations. Algebraic unnesting is desirable for cost-based algebraic optimization, because it allows such optimizers to be more readily extended with unnesting capabilities. Also it facilitates more integrated unnesting, transformation, costing, and pruning during optimization. However, the existing algebraic approaches [CM93, S95] apply to a limited subset of nested queries. In many cases, they cannot unnest sub-queries that contain collection-valued attributes. In this paper, we propose a sound and complete rule set and algorithm that unnest a significantly larger range of nested queries than the existing algebraic approaches do. We demonstrate that Magic Decorrelation, which subsumes most existing relational unnesting techniques [SPL98], can be implemented algebraically using the proposed technique.

1. Introduction

Several relational query languages, in particular SQL, allow nested queries (queries containing sub-queries). Evaluating nested queries using naive nested iteration can be very inefficient [AC75, K82]. Many unnesting techniques have been proposed to transform nested queries into more efficient forms using relational algebra or set-oriented operators [K82, GW87, D87, M92, SPL96, L96]. Kim [K82] transformed nested queries into flat queries that use joins. Ganski and Wong [GW87] discovered two problems in Kim's algorithm when handling aggregation on sub-queries. When non-equality predicates appear in sub-queries or the aggregation is COUNT, Kim's algorithm can yield incorrect results. Ganski and Wong provided a more general algorithm that uses outer-joins (instead of joins) to overcome the bugs in Kim's algorithm. Dayal [D87] unified Kim's, and Ganski and Wong's algorithms based on query-graph transformations. Also using query-graphs, Muralikrishina [M92] provided an alternative approach to fix the COUNT bug, which can be considered an extension to Ganski and Wong's algorithm. Magic Decorrelation [SPL96] decorrelates queries and their sub-queries using rewriting rules based on the QGM (Query Graph Model) of Starburst [PHH92], where magic sets [BR91] are employed to minimize the evaluation cost of decorrelated sub-queries. Kim's, Ganski and Wong's, and Dayal's algorithms are all special cases of Magic Decorrelation [SPL98].

Compared to relational queries, object queries (queries that appear in object-relational and object databases) have two new features that affect the unnesting process, namely, the occurrence of sub-queries in the *Select* clause, and correlation via collection-valued attributes (CVAs). Consider Example 1 below. It contains a sub-query in the *Select* clause, and the sub-query is correlated with the outer block via the CVA `d.Courses`. In this paper, we base our example queries on an educational record database. The database contains several types: *HighSchool*, *College*, *University*, *Course*, *Department*, *Student*, and *Professor*. The semantics of these types and their attributes should be self-evident from context. Our convention is that collections and type extents are named in plural form, for instance, *Depts*. Collection-valued attributes (CVAs) are named with the first character capitalized. For instance, `Faculty` is a CVA in *Department*. Single-valued attributes are named with lowercase characters, for instance, `title` and `advisor`. Note that example queries taken from other papers will not conform to this schema.

Example 1 Return the instructors and the textbooks of the courses offered in each department.

```
Select (d.name, A: (Select c.instructor, b.title
                    From d.Courses as c, Books as b
                    Where c.isbn = b.isbn))
From Depts as d
```

To unnest object queries, Lin and Ozsoyoglu [LO96, L97] proposed a source-to-source approach that reduces sub-queries into joins between the type extents of the correlated attributes and the collections mentioned in the outer block. Wong [W94] and Fegaras [F98] employed monoid-comprehension calculus to transform nested queries into flat algebraic expressions.

The techniques mentioned above, for both relational and object queries, can be categorized into source-to-source [K82, GW87, L97], query-graph [D87, M92, SPL96], or calculus [W94, F98] approaches. Source-to-source approaches are limited by the expressive power of query languages. Because of this limitation, they cannot unnest certain queries, for instance, those involving nested quantifiers. Source-to-source and calculus approaches cannot be readily interleaved with other transformations in algebraic optimization. One historical reason that most existing techniques are not algebraic is that they were developed before the emergence of rule-based optimizers. Another reason is that unnesting transformations cannot be easily expressed in the relational algebra or existing object algebras. For algebraic optimizers, non-algebraic unnesting has two shortcomings. First, adding unnesting functionality into an existing optimizer involves significant work, such as implementing calculus transformations. Second, an algebraic optimizer needs a separate unnesting phase in order to process nested queries, while interleaving unnesting and other transformation can often yield promising plans more quickly, especially for queries involving CVAs. For instance, an optimizer that performs unnesting between other transformations can achieve efficient expressions earlier than one that performs unnesting as a preparatory step, as we shall see in Example 5.

We remark here on the importance of finding good plans early in a cost-based algebraic optimizer framework such as Cascades (the basis for the current Microsoft SQL Server optimizer) [G95]. Unlike bottom-up optimizers, a top-down optimizer such as Cascades can use the cost of plans found so far to safely prune partial plans and sub-plans during the search process [SMB98]. The sooner a related low-cost plan is discovered, the more effective the pruning.

Cluet and Moerkotte [CM93] provided an algebraic unnesting technique for object queries. This technique, however, requires the existence of the type extents for the CVA elements to handle sub-queries that contain CVAs. Also the unnested results introduce predicates that contain set membership tests, which are typically expensive to perform.

Steenhagen [S95] provided a set of algebraic rewriting rules for unnesting object queries, based on an object algebra enhanced with some new operators, such as *nestjoin* and *markjoin*. Unnesting queries with complex quantifiers is investigated thoroughly. In many cases, nested queries with CVAs cannot be unnested. For instance, the query in Example 1 cannot be unnested using the rewriting rules provided by Steenhagen, because no transformation is available to reduce map-like operators in the presence of CVAs. In order for Steenhagen's framework to handle nested queries with CVAs, some significant transformation rules, such as those proposed in this paper, need to be introduced.

2. Algebra

We focus in this paper on the development of a sound and complete rule set and algorithm for unnesting a significant subset of nested queries. Our algebra is based on existing nested and object algebras [M77, JS82, V93, LMS93, L97]. In this section, we present the algebraic operators, and denote their semantics. Before doing so, we introduce some useful notations. Expression *L(R)* denotes the list of attributes in collection R. In the definition of an operator, symbol ← binds a list of attribute values to a list of variables. A typical use of this symbol is $L(R) \leftarrow R$, which binds the attribute values of the output of expression *R* to the variables named with the attribute labels. Symbol • is used to connect successive operators. Expression $<s_1,\ldots,s_n>$ denotes a

tuple constructor. Item s_i could be l_i:v_i, which assigns attribute labeled l_i with the value of variable v_i; s_i could also be v_i, which assigns attribute labeled v_i with the value of variable v_i. List $s_1,\ldots,s_n$ is called *projection list*. For two algebraic expressions A and B, A *depends on* B if A mentions some attributes of B, denoted as $A \prec B$; otherwise, A does not depend on B, denoted as $A \not\prec B$. Now we turn to define the operators in our algebra.

Scan $R_r = \{<r: r> \mid r \in R\}$.

Selection $\sigma^p \bullet R = \{L(R) \mid L(R) \leftarrow R, p\}$, where *p* is a predicate.

Projection $\pi^f \bullet R = \{f \mid L(R) \leftarrow R\}$, where *f* is a projection list.

Join $R \bowtie^p S = \{<L(R), L(S)> \mid L(R) \leftarrow R, L(S) \leftarrow S, p\}$.

Semi-join $R \ltimes^p S = \{L(R) \mid L(R) \leftarrow R, \exists L(S) \leftarrow S, p\}$.

Anti-join $R \triangleright^p S = \{L(R) \mid L(R) \leftarrow R, \neg\exists L(S) \leftarrow S, p\}$.

Outer-join $R \bowtie\!=^p S = (R \bowtie^p S) \cup \{<L(R), L(S):Null> \mid L(R) \leftarrow (R \triangleright^p S)\}$,

where *L(S):Null* is a shorthand for s_1:*Null*, …,s_n:*Null* , assuming *L(S)* is $\{s_1, \ldots, s_n\}$.

Unnest $\mu_{A[a]} \bullet R = \{ <L(R), a: a> \mid L(R) \leftarrow R, a \in A \}$, where $A \in L(R)$.

Nest Nest is similar to a relational group-by operator, except that the grouped elements can form new CVAs.

$$\nu_{k1,\ldots,kn}^{F= g(f)} \bullet R = \{<k_1, \ldots, k_n,, F: g(\{f \mid L(R) \leftarrow R, \forall i=1\ldots n, k_i=k_i', \exists i=1\ldots m, h_l' \neq Null\})> \mid L(R') \leftarrow R'\},$$

where $k_1,\ldots, k_n$ are the nesting key; $h_1, \ldots, h_m$ are the attributes mentioned by the projection list *f*; *g* is an aggregation function such as sum. R' is identical to *R* but with all the attributes renamed; k_i' and h_i' are the renamed k_i *and* h_i. Unnest and nest are called *restructuring operators*. They flatten or construct nested data.

D-join $R \blacksquare\!\bowtie S = \{<L(R), L(S)> \mid L(R) \leftarrow R, L(S) \leftarrow S, p\}$, where $S \prec R$.

Map $\alpha^A[g(S)]\ R = \{ <L(R), A: g(S)> \mid L(R) \leftarrow R\}$.

The map operator applies algebraic expression *S* and aggregation function *g* to every element from R. Note that map is redundant in our algebra, and can be represented using outer-djoin and nest. However, it is included to facilitate some transformations, as we shall see in Example 5.

The d-join ($\blacksquare\!\bowtie$) operator is the same as join, except that the right-hand join operand (S) depends on the left-hand one (R). Note that d-join can be reduced to regular join if the right-hand operand does not actually depend on the left-hand one, and can be reduced to unnest if the right-hand operand is a scan operator on a CVA. For instance, the expression $Depts_d \blacksquare\!\bowtie^{True} d.Students_s$ can be reduced to $\mu_{d.Students[s]} \bullet Depts_d$.

Name	Courses	
	Instructor	isbn
CS	Smith	11
	White	12
MATH	Campbell	13
	Cooper	14

Table 1. Depts

isbn	Title
12	Database
14	Algebra
11	Compiler
13	Calculus

Table 2. Books

Name	Courses		Instructor	Title
	Instructor	isbn		
CS	Smith	11	White	Database
	White	12		
MATH	Campbell	13	Cooper	Algebra
	Cooper	14		
MATH	Campbell	13	Campbell	Calculus
	Cooper	14		
CS	Smith	11	Smith	Compiler
	White	12		

Table 3. The output of R

Besides d-join (▮⋈), our algebra also includes semi-djoin(▮⋉), anti-djoin(▮▷), and outer-djoin(▮⋈=). For instance, using outer-djoin, the query in Example 1 can be expressed as

$\pi^{d.name,\ A} \bullet \nu_d{}^{A=c.instructor,\ b.title}$ R, where R is ($Depts_d$ ▮⋈= ($\pi^{c.instructor,\ b.title} \bullet$ ($d.Courses_c$ $\bowtie^{c.isbn=b.isbn}$ $Books_b$))).

Tables 1 and 2 illustrate the contents of *Depts* and *Books*. Table 3 illustrates the corresponding result *R*, a collection that contains the department name, the courses offered in the department, the instructor and textbook for each course.

3. Algebraic Unnesting

D-join was first introduced to represent sub-queries in *From* clauses [CM93]. We put d-join into a more important use in representing, transforming and eventually unnesting nested queries. To represent nested queries, most object and object-relational algebras use map-like operators. Example of such operators are the extended selection and projection in the nested algebra used in the DASDBS project [JS82], and the apply operator in the AQUA object algebra [LMS93]. Map-like operators do not support flexible transformations, because they do not commute with other operators such as join, projection, and selection. We use d-join to overcome this problem. D-join is higher-ordered, thus is more expressive than relational operators, and furthermore permits more transformations than map-like operators. The idea of our unnesting approach is to first represent a query using an expression that consists of d-join, relational, and restructuring operators, then push down d-joins to the bottom of the expression tree. When the right-hand operand of a d-join becomes a scan operator, the d-join can be reduced to a join or unnest. Below, we describe the steps taken to unnest a given expression.

Step One: Normalization

The unnesting process can start with any expression expressible in the algebra. First, the anti-djoin, semi-djoin, and outer-djoin operators in the expression are rewritten into d-joins using *normalization* rules, which are given in Appendix B. Normalization produces expressions that consist of d-join, relational, and restructuring operators. Such expressions allow extensive transformations. Below are several example normalization rules. Note that, in the following rules, *R* and *S* could be expressions, or scan operators over base collections or CVAs.

Rule 1 (map elimination) $\alpha^A[g(S)]\ R \equiv \nu_{L(R)}{}^{A=g(L(S))} \bullet (\ R$ ▮⋈=^{True} S),

where *g* denotes an aggregation function.

Rule 2 (outer-djoin elimination) R ▮⋈=^{p} S ≡ R ▮⋈^{p} S.

The condition for Rule 2 is that reducing outer-djoin to d-join will not affect the correctness of the succeeding operators. Such condition can be captured by an inherited property propagated down from the top of an expression. For instance, in the expression

$\sigma^{a>0} \bullet \nu_d{}^{a=count(s)} \bullet$ ($Depts_d$ ▮⋈=$^{s.age>18}$ $d.Students_s$),

the outer-djoin (▮⋈=) can be safely reduced to d-join (▮⋈), since ruling out the departments with no old students does not affect the result of the succeeding selection (σ). In case the outer-djoin can not be reduced by Rule 2, it will be rewritten into d-join and regular outer-join using Rule 3.

Rule 3 (outer-djoin elimination) R ▮⋈=^{p} S ≡ $\pi^{L(R):L(R'),\ L(S)} \bullet$ (R′ ⋈=$^{L(R')=L(R)}$ (R ▮⋈^{p} S)),

where *R′* is identical to *R* but with all the attributes renamed. Let *L(R)* be *{r_1, …, r_n}*, and *L(R′)* be *{r_1′, …, r_n′}*. Then *L(R):L(*R′*)* is a shorthand for *r_1:r_1′,…, r_n:r_n′*, and *L(R)=L(R)* is a shorthand for (*r_1=r_1′)*∧…∧*(r_n=r_n′)*.

Step Two: Operand Reduction

The second step is to push down the d-join operators systematically using *operand reduction* rules as well as join reordering rules [JK84]. The operand reduction rules are listed in Appendix B. Here, we give some example rules. In the definition of a rule, $p(A_1, \ldots, A_i)$ denotes a predicate p over the attributes in collections $A_1,\ldots$, and A_i. Items p, q, and r represent the predicates before transformation. Items x, y, and z represent the predicates after transformation; x, y, and z are derived from p, q, and r such that $x \wedge y \wedge z$ is equivalent to $p \wedge q \wedge r$.

Rule 4 (right-hand operand reduction) $(R \bowtie^{p(R,S)} S) \blacktriangleright\bowtie^{q(R,S,T)} T \equiv R \bowtie^{x(R,S,T)} (S \blacktriangleright\bowtie^{y(S,T)} T)$, where $T \not\prec R$.

Rule 5 (left-hand operand reduction) $R \blacktriangleright\bowtie^{p(R,S,T)} (S \bowtie^{q(S,T)} T) \equiv (R \blacktriangleright\bowtie^{x(R,S)} S) \bowtie^{y(R,S,T)} T$, where $T \not\prec R$.

Rule 6 (both operand reduction)

$(R \bowtie^{p(R,S)} S) \blacktriangleright\bowtie^{q(R,S,U,V)} (U \bowtie^{r(U,V)} V) \equiv (R \blacktriangleright\bowtie^{x(R,U)} U) \bowtie^{y(R,S,U,V)} (S \blacktriangleright\bowtie^{z(S,V)} V)$, where $U \not\prec S$ and $V \not\prec R$.

Rules 4, 5, and 6 reduce the number of operators in the operands of d-joins: Rule 4 reduces left-hand operands; Rule 5 reduces right-hand operands; Rule 6 reduces both operands. Evidently, in order to fire these rules, join reordering rules have to be first applied to either operand or both operands. Operands are reduced by pushing down d-join through joins. Other operand reduction rules can push down d-join through various operators such as unary operators, restructuring operators, semi-joins, anti-joins, and outer-joins.

Step Three: D-join Reduction

When a d-join is pushed down to the bottom of the algebraic tree, its right-hand operand will be a scan operator. It will be reduced to join or select and unnest operators, using *d-join reduction* rules, Rule 7. After all d-joins are reduced, an unnested expression will result.

Rule 7 (d-join reduction)

7.1. $R \blacktriangleright\bowtie^{p} S \equiv (R \bowtie^{p} S)$, where $S \not\prec R$.

7.2. $R \blacktriangleright\bowtie^{p} A_a \equiv \sigma^{p} \bullet \mu_{A[a]} \bullet R$, where $A \in L(R)$.

Example 2 Return student and program pairs; in each pair, the student must have successfully taken more than five core courses required by the program.

```
Select s, p
From Students as s, Programs as p
Where 5 < Count (Select c
                 From s.Transcript as t, p.Core as c
                 Where (t.title = c.title) and (t.grade <'F'))
```

The query is initially represented as (Figure 1a):

$\pi^{s,p} \bullet \sigma^{5<a} \bullet \alpha^{a}$ [count $((\sigma^{t.grade <'F'} \bullet s.Transcript_t) \bowtie^{t.title=c.title} Core_c)$] $(Students_s \bowtie^{True} Programs_p)$.

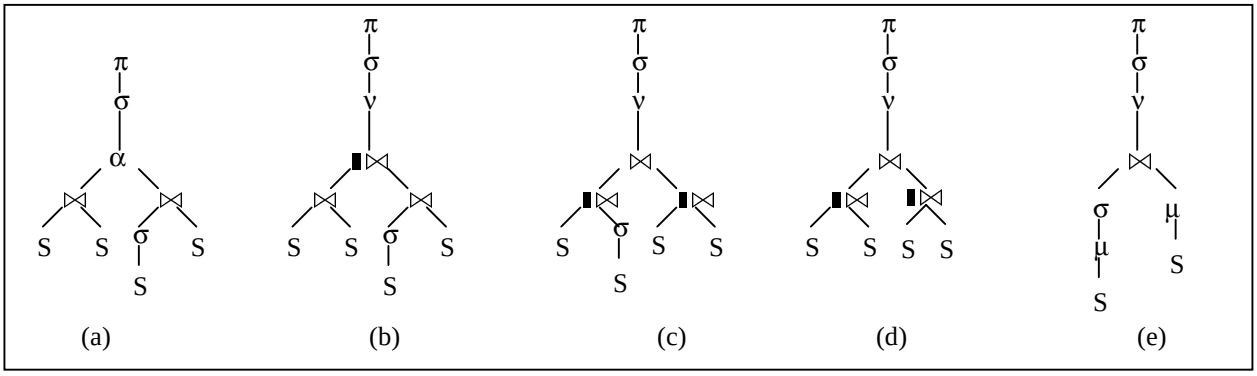

Figure 1. Unnesting Procedure for Example 2.

Figure 1 illustrates the algebraic tree representation of the unnesting process, where the leaf nodes (*S*'s) represent the scan operators over base collections or CVAs. The figure demonstrates how d-joins are pushed down and eventually reduced. To unnest the expression above, the map operator (α) is first rewritten into outer-djoin ($\blacktriangleright\!\bowtie\!=$), which is then reduced to d-join ($\blacktriangleright\!\bowtie$) by the normalization rules, as demonstrated by Figure 1b:

$$\pi^{s,p} \bullet \sigma^{5<a} \bullet \nu_{s,p}{}^{a=count(t,c)} \bullet ((Students_s \bowtie^{True} Programs_p) \blacktriangleright\!\bowtie^{True} ((\sigma^{t.grade <'F'} \bullet s.Transcript_t) \bowtie^{t.title=c.title} p.Core_c)).$$

Second, Rule 6 is applied to reduce both operands of the d-join operator, then the left d-join operator ($\blacktriangleright\!\bowtie$) is pushed down and combined with the selection, as demonstrated by Figures 1c and 1d:

$$\pi^{s,p} \bullet \sigma^{5<a} \bullet \nu_{s,p}{}^{a=count(t,c)} \bullet ((Students_s \blacktriangleright\!\bowtie^{True} (\sigma^{t.grade <'F'} \bullet s.Transcript_t) \bowtie^{t.title=c.title} (Programs_p \blacktriangleright\!\bowtie^{True} p.Core_c))$$
$$\equiv \pi^{s,p} \bullet \sigma^{5<a} \bullet \nu_{s,p}{}^{a=count(t,c)} \bullet (Students_s \blacktriangleright\!\bowtie^{t.grade <'F'} s.Transcript_t) \bowtie^{t.title=c.title} (Programs_p \blacktriangleright\!\bowtie^{True} p.Core_c)).$$

Third, Rule 7 is applied to reduce d-joins, as demonstrated by Figure 1e:

$$\pi^{s,p} \bullet \sigma^{5<a} \bullet \nu_{s,p}{}^{a=count(t,c)} \bullet ((\sigma^{t.grade <'F'} \bullet \mu_{s.Transcript[t]} \bullet Students_s) \bowtie^{t.title=c.title} (\mu_{p.Core[c]} \bullet Programs_p)).$$

Our unnesting algorithm employs many transformation rules, because the algorithm has to deal with a large variety of operators. Despite the relatively large rule set, the number of rules applicable to a given pattern or expression during unnesting is quite limited. Also these rules can be applied in a deterministic manner. Therefore, the complexity of rule control and the search space is moderate.

Theorem The unnesting rule set and algorithm described above are sound and complete for the nested queries that are expressible in the algebra defined in Section 2.

Sketch of the proof

The soundness of the rules can be demonstrated with straightforward set-theoretic reasoning. An example proof is given in Appendix A. The completeness is proved by showing that the unnesting algorithm will unnest any expression expressible in the algebra. For a given expression, Step One generates a normalized expression where d-joins ($\blacktriangleright\!\bowtie$) are the only higher-ordered operators.

Step Two reduces the operands of all the d-join operators, such that the only operators left in their right-hand operands are scan operators. Step Two starts with an expression obtained from Step One. If the expression contain d-joins, there is at least one d-join operator that does not contain d-joins in its operands. For such an operator, the operand reduction and join-reordering rules are applied repeatedly. Once join-reordering rules are applied, at least one operand reduction rule will apply to the d-join, because the operand reduction rules cover all the possibilities. The right-hand operand of the d-join will be eventually reduced to a scan operator.

Step Three transforms the d-joins with only scan operators in their right-hand operands into join or unnest operators. Repeating Steps Two and Three will eventually eliminate all the d-joins in a normalized expression produced by Step One. ■

Most nested relational and object queries are expressible in our algebra. Therefore, the proposed unnesting technique is complete for a rather large subset of nested queries. However, our algebra does not include all the operators required for processing query languages such as OQL [C97]. Language features such as method invocation, union, intersection, and multiple collection types are not yet supported. Many of these features can be handled by adding new operators into the algebra, and have no significant impact on the query unnesting algorithm presented here.

4. Evaluation

4.1 Magic Decorrelation as an Application

Magic Decorrelation [SPL96] considers correlated sub-queries as parametric queries, and decorrelates these sub-queries by joining them with a *magic set* (the set of values for a parameter). The decorrelation rewriting is based on query graphs (QGM). Other unnesting techniques such as Kim's, Ganski and Wong's, and Dayal's are special cases of Magic Decorrelation [SPL98]. Magic Decorrelation is implemented in the optimizer of IBM's DB2 Universal Database system. Due to incompatibility with the DB2 optimizer framework, Magic Decorrelation has to be implemented as an independent rewriting component. Below, we illustrate Magic Decorrelation using Example 3, an example drawn from the original Magic Decorrelation paper [SPL96].

Example 3 Find young employees who are paid more than the average salary in their department.

```
Select *
From Emps e
Where  (e.age < 30) and (e.sal > Avg  (Select e1.sal
                                       From Emps e1
                                       Where e1.dept = e.dept))
```

Using Kim's (or Cluet and Moerkotte's) approach, the query will be unnested into

$$\pi^{e} \bullet ((\sigma^{e.age<30} \bullet Emp_e) \bowtie^{e.sal>a \wedge e1.dept=e.dept} (\nu_{e1.dept}^{a=avg(e1.sal)} \bullet Emp_{e1})). \quad (3.1)$$

Observing that the sub-expression $(\nu_{e1.dept}^{a=avg(e1.sal)} \bullet Emp_{e1})$ may compute average salary for departments that do not even have young employees, the Magic Decorrelation approach will unnest the query into a more efficient form:

$$\pi^{e} \bullet ((\sigma^{e.age<30} \bullet Emp_e) \bowtie^{e.sal>a \wedge e.dept=e1.dept} (\nu_{e1.dept}^{a=avg(e1.sal)} \bullet (Emp_{e1} \bowtie^{e1.dept=e.dept} (\pi^{e.dept} \bullet \sigma^{e.age<30} \bullet Emp_e)))). \quad (3.2)$$

The result of the sub-expression $(\pi^{e.dept} \bullet \sigma^{e.age<30} \bullet Emp_e)$ is the magic set [BR91] used for decorrelation, representing the set of departments with young employees. The advantage of Expression (3.2) is that it computes the average salary for a department only when necessary. The fewer departments that have young employees, the more efficient Expression (3.2) is compared to Expression (3.1).

We observe that Magic Decorrelation is an application of our unnesting approach. Implementing Magic Decorrelation in an algebraic fashion, rather than as a separate module, will allow it to participate in extensive costing and search space exploration in a cost-based algebraic optimizer. To realize Magic Decorrelation, we introduce a new transformation rule called the magic rule:

Rule 8 (magic) $R \mathbin{▮\!\bowtie^{p}} S \equiv \pi^{L(R):L(R'),\, L(S)} \bullet (R' \bowtie^{a'=a} ((\pi^{a} \bullet R) \mathbin{▮\!\bowtie^{p}} S))$,

where *a* is the attribute via which *S* depends on *R*; *R´*is identical to *R* but with all the attributes renamed; *a´*is the renamed *a*. This rule generates the magic-set $(\pi^{a} \bullet R)$. Further transformations will relocate the magic-set to the most beneficial place. Consider Example 3. The query is initially represented as

$$\pi^{e} \bullet ((\sigma^{e.age<30} \bullet Emp_e) \mathbin{▮\!\bowtie^{e.sal>a}} (\nu_{e1.dept}^{a=avg(e1.sal)} \bullet \sigma^{e1.dept=e.dept} \bullet Emp_{e1})).$$

Applying the magic rule will give

$$\pi^{e} \bullet \pi^{e:e'} \bullet ((\sigma^{e'.age<30} \bullet Emp_{e'}) \bowtie^{e'.dept=e.dept} ((\pi^{e.dept} \bullet \sigma^{e.age<30} \bullet Emp_e) \mathbin{▮\!\bowtie^{e.sal>a}} (\nu_{e1.dept}^{a=avg(e1.sal)} \bullet \sigma^{e1.dept=e.dept} \bullet Emp_{e1}))).$$

Combine the two projection operators, and reducing the right-hand operand of the d-join yields

$$\pi^{e:e'} \bullet (\sigma^{e'.age<30} \bullet Emp_{e'}) \bowtie^{e'.dept=e1.dept} (\nu_{e.dept,\, e1.dept}^{a=avg(e1.sal)} \bullet ((\pi^{e.dept} \bullet \sigma^{e.age<30} \bullet Emp_e) \mathbin{▮\!\bowtie^{e.sal>a \wedge e1.dept=e.dept}} Emp_{e1})).$$

Reducing the d-join operator gives an expression that is essentially Expression (3.2).

4.2 Comparison to Other Unnesting Techniques

We contrast our unnesting approach with the existing ones in two aspects. Table 4 compares the range of queries that our approach and the other two algebraic approaches can handle. Table 5 and 6 indicates the number of transformation steps that our approach and other approaches take to derive the same unnested expressions.

In Table 4, *WMS* represents our approach; *CM* represents Cluet and Moerkotte's approach; *Steen* represents Steenhagen's approach. Four types of queries are considered: Types J, JA, J-CVA, and JA-CVA. Types J and JA [CM93] are nested queries that have an inner block depending on the outer block. In Type J, the inner block returns a set. In Type JA, the inner block returns a single element. Type J-CVA (Type JA-CVA) is the same as Type J (Type JA) except that the inner block mentions some CVAs of the collections that belong to the outer block. For Types J-CVA and JA-CVA, as mentioned in Section 1, Cluet and Moerkotte's approach cannot unnest queries of both types, when appropriate type extents are not available. Steenhagen's approach cannot handle these types either, for lack of appropriate transformation rules.

	WMS	CM	Steen
J	Yes	Yes	Yes
JA	Yes	Yes	Yes
J-CVA	Yes	No	No
JA-CVA	Yes	No	No

Table 4. Contrast the algebraic unnesting techniques

	WMS	Magic
Example 3	4	9
Query (1)	7	17
Query (2)	4	9
Query (3)	7	10

Table 5. Our technique vs. Magic Decorrelation

	WMS	Fegaras
Example 1	3	3
Example 2	3	5
Example 3	3	3
Example 4	5	8

Table 6. Our technique vs. Fegaras's

Table 5 demonstrates the number of transformation steps that our approach and Magic Decorrelation take to unnest a query. Example 3 is the previous example. Queries (1), (2) and (3) are drawn from the original paper on Magic Decorrelation [SPL96]. In general, Magic Decorrelation takes more transformation steps than our approach, especially when the inner blocks have aggregations. Each aggregation block in a QGM graph requires at least three transformation steps to decorrelate, while in algebraic unnesting, a nest operator can be decorrelated with one transformation (pushing down d-join through nest).

Table 6 contrasts the number of transformation steps that our approach and Fegaras's approach take to derive the same unnested expressions. The first three queries are from the previous examples. The fourth one is from Example 4 below. The two approaches yield the same unnested results for some queries, for instance, Example 1 and 3. However, the unnested results could be different, for instance, for queries in Examples 2 and 4. We notice that our approach often yields more efficient expressions, especially, for queries that contain multiple CVAs in the sub-queries. Therefore, to derive the same efficient unnested expressions for such queries, Fegaras's approach requires more steps. We illustrate using Example 4.

Example 4 For each department, find the professors who are not advising old students.

```
Select (d, F: (Select f
               From d.Faculty as f
               Where not exists s in d.Students: (s.advisor=f.name) and (s.age>30)))
From Depts as d
```

Fegaras's algorithm takes five steps to unnest this query into

$$\pi^{d,F} \bullet \Gamma_d{}^{F=\cup/k=False} \bullet \Gamma^{k=\vee/\,(s.advisor=f.name \wedge s.age>30)} \bullet \mu_{d.Students[s]} \bullet \mu_{d.Faculty[f]} \bullet Depts_d, \qquad (4.1)$$

where operator Γ, defined in Fegaras's original paper [F98] is a generalized relational grouping operator that allows comprehension accumulators such as disjunction (∧), and set union (∪) to be applied to grouped elements. Our unnesting algorithm takes five steps to unnest the query into an different expression

$$\pi^{d,F} \bullet \nu_d^{F=f} \bullet ((\mu_{d.Faculty[f]} \bullet Depts_d) \triangleright^{d=d' \wedge f.name=s.advisor} (\sigma^{s.age>30} \bullet \mu_{d'.Students[s]} \bullet Depts_{d'})). \quad (4.2)$$

Expression (4.2) is likely to be more efficient than (4.1), due to smaller intermediate results. It would take Fegaras's approach three steps to rewrite (4.2) into (4.1): transform $Depts_d$ into $Depts_d \bowtie Depts_{d'}$; push down the two unnest operators; combine Γ and ⋈ into ▷.

4.3 Transformation Advantage

A particular advantage of algebraic unnesting is to allow the unnesting rules to be applied among other transformations during algebraic optimization. Sophisticated techniques such as region-based optimization [AMP93, MDZ93, ONP95] can utilize this capability to search optimal expressions more effectively. We use Example 5 to illustrate that a simple strategy, performing unnesting between other transformations, can help generate good expressions earlier than performing unnesting as a preparatory step.

Example 5 The following query returns triples of a high school, a college, and a university, where the high school has graduates who entered the college, and the college has graduates who entered the university.

Select s, c, u
From Schools as s, Colleges as c, Universities as u
Where Exists (Select *
From s.Graduates as g_1, c.Students as s_1
Where g_1.ssn = s_1.ssn) and Exists (Select *
From c.Graduates as g_2, u.Students as s_2
Where g_2.ssn = s_2.ssn)

Using our algebra, this query is represented as

$$(Schools_s \bowtie Colleges_c \bowtie Universities_u) \ltimes (s.Graduates_{g1} \bowtie^{g1.ssn=s1.ssn} c.Students_{s1}) \ltimes (c.Graduates_{g2} \bowtie^{g2.ssn=s2.ssn} u.Students_{s2}). \quad (5.1)$$

Before our unnesting transformation is applied, the two semi-djoins could be reordered using the communitivity rule $(R \ltimes S) \ltimes T \equiv (R \ltimes T) \ltimes S$:

$$(Schools_s \bowtie Colleges_c \bowtie Universities_u) \ltimes^{g2.ssn=s2.ssn} (c.Graduates_{g2} \bowtie u.Students_{s2}) \ltimes^{g1.ssn=s1.ssn} (s.Graduates_{g1} \bowtie c.Students_{s1}). \quad (5.2)$$

The purpose of this transformation is to push down the more restrictive semi-djoin. The query can then be unnested using the proposed unnesting algorithm into:

$$\pi^{h,c,s} \bullet ((\mu_{c.Students[s1]} \bullet ((\mu_{c.Graduates[g2]} \bullet Colleges_c) \bowtie^{g2.ssn=s2.ssn} (\mu_{u.Students[s2]} \bullet Universities_u))) \bowtie^{g1.ssn=s1.ssn} (\mu_{s.Graduates[g1]} \bullet Schools_s)). \quad (5.3)$$

Expression (5.3) is optimal, assuming that, in the original expression, the second semi-djoin is more restrictive than the first one. Without interleaving unnesting with other transformations, a traditional optimizer will probably first unnest [F98] the query into

$$\pi^{h,c,s} \bullet \sigma^{g2.ssn = s2.ssn} \bullet \mu_{c.Graduates[g2]} \bullet \mu_{u.Students[s2]} \bullet \sigma^{g1.ssn = s1.ssn} \bullet \mu_{c.Students[s1]} \bullet \mu_{s.Graduates[g1]} \bullet (Schools_s \bowtie Colleges_c \bowtie Universities_u), \quad (5.4)$$

then perform further transformations and planning. The preferred expression (5.3) can be derived eventually, but with much more effort. Here is a possible transformation process: all the unnest (μ) operators are pushed down to the bottom of the expression tree; both selections are pushed down and merged into joins; joins are reordered; the unnest operator, $\mu_{c.Students[s1]}$, is pulled up.

The example suggests that, to derive the same unnested expressions for a given nested query, an optimizer that does not interleave unnesting and other transformations may need more steps than one that does.

5. Conclusion

For a cost-based algebraic optimizer, algebraic unnesting techniques have several advantages. First, adding unnesting capacity to the optimizer can be easily achieved by adding some unnesting rules. Second, interleaving unnesting and other transformations often helps generate good expressions early. Since a cost-based optimizer relies on the costs of available expressions to prune expensive expressions, generating good expressions earlier will make the optimization more effective. However, most existing unnesting techniques [K82, GW87, D87, M92, L96, SPL96, F98] are not algebraic. In general, the existing algebraic approaches [CM93, S95] cannot handle nested queries with collection-valued attributes. To overcome this problem, we provide a sound and complete rule set and algorithm that unnest a significantly larger subset of object queries than the existing algebraic approaches do. In addition, we show that Magic Decorrelation, a query graph-based technique that generalizes many other unnesting techniques, can be realized algebraically with the proposed technique. Hence, our technique subsumes existing relational unnesting techniques.

One might wonder why there is no algebraic technique for relational queries, since our technique can handle them. The answer is that our technique uses non-relational (higher-ordered) operators to unnest relational expressions – it is not expressible using just relational operators. Hence, the proposed technique is also interesting to plain relational systems.

Currently, we are implementing unnesting as part of a cost-based algebraic optimizer for CVA queries, in the Columbia optimizer framework [SMB98]. In the future, we will investigate the interaction between unnesting and particular search strategies. For instance, in the Cascade optimization framework [G95], upon which the Columbia framework is based, the "promise" property of a transformation rule guides the optimizer to choose the best rule to fire when several rules are applicable to a given pattern. It would be interesting to see how the "promise" mechanism works for nested object query optimization. Note that two unnesting rules can have different promises. For instance, Rule 6 (Section 3) should be assigned higher promise than Rule 5, because it reduces the operands more aggressively (reduces both operands). Eventually, we will develop and validate experimentally a search engine powered by seamless integration of unnesting and cost-based optimization.

References

[AC75] M. Astrahan, D. Chamberlin. Implementation of a Structured English Query Language, *Comm. of the ACM*, Vol. 18, No. 10, 1975.

[AH88] S. Abiteboul, R.Hull. Restructuring Hierarchical Database Objects. *TCS* 62(1-2), 1988.

[AMP93] A. Kemper, G. Moerkotte, K. Peithner. A Blackboard Architecture for Query Optimization in Object Bases. *VLDB 1993*.

[BR91] C. Beeri, R. Ramakrishnan. On the Power of Magic. *Journal of Logic Programming*, 10-255, 1991.

[B90] J. V. D. Bussche. A Formal Basis for Extending SQL to Object-Oriented Databases. *Bulletin of the EATCS*, Vol. 40, 1990.

[C97] R. G. G. Cattell. The Object Database Standard: ODMG-93, Release 2.0, Morgan Kaufmann, 1997.

[CM93] S. Cluet, G. Moerkotte. Nested Queries in Object Bases. *DBPL 1993*.

[D87] U. Dayal, Of Nests and Trees: A Unified Approach to Processing Queries That Contain Nested Sub-queries, Aggregates and Quantifiers. *VLDB* 1987.

[F98] L. Fegaras. Query Unnesting in Object-Oriented Databases. *ACM SIGMOD* 1998.

[G95] G. Graefe. The Cascades Framework for Query Optimization. *Data Engineering Bulletin* 18(3), 1995.

[GBC98] G. Graefe, R. Bunker, S. Cooper. Hash Joins and Hash Teams in Microsoft SQL Server. *VLDB 1998*.

[GW87] R. Ganski, H. K. T. Wong, Optimization of Nested SQL Queries Revisited, *ACM SIGMOD 1987*.

[J82] J. D. Ullman. Query Processing in Universal Relation Systems. *Database Engineering Bulletin* 5(3),

1982.
[J88] J. D. Ullman. Principles of Database and Knowledge Systems. Computer Science Press, 1988.
[JK84] M. Jarke, J. Koch. Query Optimization in Database System. *ACM Computer Survey*, June 1984.
[JS82] G. Jaeschke, H. J. Schek. Remarks on the Algebra of Non-First-Normal-Form Relations, *PODS* 1982.
[K82] W. Kim, On Optimizing an SQL-like Nested Query, *ACM Trans. on Database Systems*, Sept. 1982.
[L97] J. Lin. Processing and Optimizing OODB Queries by O-Algebra, Ph.D. Dissertation, Case Western Reserve University, 1997.
[LM96] J. Lin, Z. M. Özsoyoglu. Processing OODB Queries by O-Algebra. *CIKM 1996*.
[LMS93] T. Leung, G. Mitchell, B. Subramanian, B. Vance, S. L. Vandenberg, S. B. Zdonik. The AQUA Data Model and Algebra. *DBPL 1993*.
[M77] A. Makinouchi. A Consideration of Normal Form of Not-Necessarily-Normalized Relations in the Relational Data Model. *VLDB 1977*.
[M92] M. Muralikrishna, Improved Unnesting Algorithms for Join Aggregate SQL Queries, *VLDB 1992*.
[MDZ93] G. Mitchell, U. Dayal, S. B. Zdonik. Control of an Extensible Query Optimizer: A Planning Based Approach. *VLDB 1993*.
[MP94] I. S. Mumick, H. Priahesh. Implementation of Magic-Sets in Starburst. *SIGMOD 1994*.
[ONP95] F. Ozcan, S. Nural, P. Koksal, M. Altinel, A. Dogac. A Region Based Query Optimizer Through Cascades Query Optimizer Framework. *Data Engineering Bulletin* 18(3), 1995.
[OOM87]G. Ozsoyoglu, Z. M. Özsoyoglu, V.Matos. Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions. *TODS* 12(4), 1987.
[PHH92] H. Pirahesh, J. M. Hellerstein, W. Hasan. Extensible/Rule Based Query Rewrite Optimization in Starburst. *ACM SIGMOD 1992*.
[S95] H. J. Steenhagen. Optimization of Object Query Language, PH.D thesis, University of Twente.
[SMB98] L. Shapiro, D. Maier, K. Billings, Y. Fan, B. Vance, Q. Wang, H. Wu. Safe pruning in the Columbia query optimizer, www.cs.pdx.edu/~len/pruning.doc or pruning.doc.zip.
[SPL96] P. Seshadri, H. Pirahesh, T. Y. C. Leung, Complex Query Decorrelation, *Proc. the 20th Intl. Conference on Data Engineering*, March, 1996.
[SPL98] P. Seshadri, H. Pirahesh and T. Leung, Query Processing Techniques for Correlated Queries, *IBM Research Report*, August 17, 1998.
[V93] S. Vandenberg. Algebras for Object-Oriented Query Languages. Ph.D. Dissertation, University of Wisconsin-Madison, 1993.
[W94] L. Wong. Querying Nested Collections. PhD Thesis, University of Pennsylvania, 1994.

Appendix A. Correctness of the Unnesting Rules

The correctness of the unnesting rules is proved using set theoretic reasoning. For illustration, we sketch the proof for Rule 3 of Section 3 below.

$\pi^{L(R):L(R'),\,L(S)} \bullet (R' \bowtie^{=}{}^{L(R')=L(R)} (R \blacksquare\bowtie^{p} S))$

$\equiv \pi^{L(R):L(R'),\,L(S)} \bullet (R' \bowtie^{L(R')=L(R)} (R \blacksquare\bowtie^{p} S)) \cup$

$\pi^{L(R):L(R'),\,L(S)} \bullet \{<L(R'), L(R)\text{: Null}, L(S)\text{: Null}> \mid L(R') \, L \leftarrow (R' \triangleright^{L(R')=L(R)} (R \blacksquare\bowtie^{p} S))\}$

$\equiv \pi^{L(R):L(R'),\,L(S)} \bullet \{<L(R'), L(R), L(S)>$

$\mid L(R') \leftarrow R', (L(R), L(S)) \leftarrow \{ <(L(R), L(S)> \mid L(R) \leftarrow R, L(S) \leftarrow S, p\}, L(R')=L(R)\} \cup$

$\pi^{L(R):L(R'),\,L(S)} \bullet \{<L(R'), L(R)\text{: Null}, L(S)\text{: Null}>$

$\mid L(R') \leftarrow R', \neg\exists (L(R), L(S)) \leftarrow \{ <L(R), L(S)> \mid L(R) \leftarrow R, L(S) \leftarrow S, p\}, L(R') = L(R)\}$

$\equiv \{< L(R)\text{: } L(R'), L(S)> \mid L(R') \leftarrow R', L(R) \leftarrow R, L(S) \leftarrow S, p, L(R') = L(R)\} \cup$

$\{< L(R)\text{: } L(R'), L(S)\text{: Null}> \mid L(R') \leftarrow R', L(R) \leftarrow R, \neg\exists L(S) \leftarrow \{ L(S) \mid L(S) \leftarrow S, p\}, L(R') = L(R)\}$

$\equiv \{<L(R), L(S)> \mid L(R) \leftarrow R, L(S) \leftarrow S, p\} \cup \{\{< L(R), L(S)\text{: Null}> \mid L(R) \leftarrow R, \neg\exists L(S) \leftarrow S, p\}$

$\equiv R \blacksquare\bowtie^{=}{}^{p} S.\blacksquare$

Appendix B. Unnesting Transformation Rules

There are three kinds of transformation rules involved in query unnesting: the normalization rules, the operand reduction rules, and the d-join reduction rules. Note that the appendix includes the rules appear in Section 3 and 4. However, the rule numbers assigned here are irrelevant to those in Section 3 and 4.

B.1. The Normalization Rules

Rule 1 (map elimination) $\alpha^{A}[\, g(S)\,]\, R \equiv \nu^{A=g(L(S))}_{L(R)} \bullet (\, R \,\blacktriangleright\!\bowtie\!=^{True}\, S)$.

Rule 2 (semi-djoin elimination) $R \,\blacktriangleright\!\ltimes^{p}\, S \equiv \pi^{L(R)} \bullet (R \,\blacktriangleright\!\bowtie^{p}\, S)$.

Rule 3 (anti-djoin elimination) $R \,\blacktriangleright\!\triangleright^{p}\, S \equiv \pi^{L(R):L(R')} \bullet (R' \triangleright^{L(R')=L(R)} \bullet (R \,\blacktriangleright\!\bowtie^{p}\, S))$.

Rule 4 (outer-djoin elimination)

4.1. $R \,\blacktriangleright\!\bowtie\!=^{p}\, S \equiv R \,\blacktriangleright\!\bowtie^{p}\, S$, where $S \not\prec R$.

4.2. $R \,\blacktriangleright\!\bowtie\!=^{p}\, S \equiv \pi^{L(R):L(R'),\, L(S)} \bullet (R' \bowtie\!=^{L(R')=L(R)} (R \,\blacktriangleright\!\bowtie^{p}\, S))$.

B.2. The Operand Reduction Rules

Rule 5 (d-join through join)

5.1. $(R \bowtie^{p(R,S)} S) \,\blacktriangleright\!\bowtie^{q(R,S,T)}\, T \equiv R \bowtie^{x(R,S,T)} (S \,\blacktriangleright\!\bowtie^{y(S,T)}\, T)$, where $T \not\prec R$.

5.2. $R \,\blacktriangleright\!\bowtie^{p(R,S,T)} (S \bowtie^{q(S,T)} T) \equiv (R \,\blacktriangleright\!\bowtie^{x(R,S)} S) \bowtie^{y(R,S,T)} T$, where $T \not\prec R$.

5.3. $R \,\blacktriangleright\!\bowtie^{p(R,S,T)} (S \bowtie^{q(S,T)} T) \equiv (R \,\blacktriangleright\!\bowtie^{x(R,S)} S) \,\blacktriangleright\!\bowtie^{y(R,S,T)}\, T$, where $S \prec R$ and $T \prec R$.,

5.4. $(R \bowtie^{p(R,\, S)} S) \,\blacktriangleright\!\bowtie^{q(R,S,U,V)} (U \bowtie^{r(U,V)} V) \equiv (R \,\blacktriangleright\!\bowtie^{x(R,U)} U) \bowtie^{y(R,S,U,V)} (S \,\blacktriangleright\!\bowtie^{z(S,V)}\, V)$, where $U \not\prec S$ and $V \not\prec R$.

Rule 6 (d-join through semi-join)

6.1. $R \,\blacktriangleright\!\bowtie^{p} (S \ltimes^{q} T) \equiv (R \,\blacktriangleright\!\bowtie^{p} S) \ltimes^{q} T$, where $T \not\prec R$.

6.2. $R \,\blacktriangleright\!\bowtie^{p(R,S,T)} (S \ltimes^{q(R,S,T)} T) \equiv (R \,\blacktriangleright\!\bowtie^{x(R,S)} S) \ltimes^{y(R,S,T') \wedge (L(R)=L(R'))} (R' \,\blacktriangleright\!\bowtie^{z(R',T')} T')$.

6.3. $(R \ltimes^{p} S) \,\blacktriangleright\!\bowtie^{q} T \equiv (R \,\blacktriangleright\!\bowtie^{q} T) \ltimes^{q} S$.

In Rule 6.2, R' is identical to R but with all the attributes renamed; T' is identical T except that the attributes of R that appear in T are renamed according to the change made in R'.

Rule 7 (d-join through anti-join) The rules for pushing d-join down through anti-join are similar to Rule 6.

Rule 8 (d-join through outer-join)

8.1. $R \,\blacktriangleright\!\bowtie^{p} (S \bowtie\!=^{q} T) \equiv (R \,\blacktriangleright\!\bowtie^{p} S) \bowtie\!=^{q} T$, where $T \not\prec R$.

8.2. $R \,\blacktriangleright\!\bowtie^{p(R,S,T)} (S \bowtie\!=^{q(R,S,T)} T) \equiv \pi^{L(R),L(S),\, L(T):L(T')} \bullet ((R \,\blacktriangleright\!\bowtie^{x(R,S)} S) \bowtie\!=^{y(R,S,T') \wedge (L(R)=L(R'))} (R' \,\blacktriangleright\!\bowtie^{z(R',T')} T'))$.

In Rule 8.2, R' is identical to R but with all the attributes renamed; T' is identical T except that the attributes of R that appear in T are renamed according to the change made in R'.

Rule 9 (d-join through unary operators)

9.1. $R \,\blacktriangleright\!\bowtie^{p} (\pi^{f} \bullet S) \equiv \pi^{L(R),\, f} \bullet (R \,\blacktriangleright\!\bowtie^{p} S)$.

9.2. $(\pi^{f} \bullet R) \,\blacktriangleright\!\bowtie^{p} S \equiv \pi^{f,\, L(s)} \bullet (R \,\blacktriangleright\!\bowtie^{p} S)$.

9.3. $R \,\blacktriangleright\!\bowtie^{p} (\sigma^{q} \bullet S) \equiv R \,\blacktriangleright\!\bowtie^{p \wedge q} S$.

9.4. $(\sigma^{p} \bullet R) \blacktriangleright\!\bowtie^{q} S \equiv R \blacktriangleright\!\bowtie^{p \wedge q} S$.

Rule 10 (d-join through restructuring operators) The following rules push down d-join through the restructuring operator in its right-hand operand. The rules for pushing down d-join to its left-hand operand can be similarly derived.

10.1. $R \blacktriangleright\!\bowtie^{p(R,S)} (\mu_{A[a]} \bullet S) \equiv \mu_{A[a]} \bullet (R \blacktriangleright\!\bowtie^{p(R,S)} S)$.

10.2. $R \blacktriangleright\!\bowtie^{p(R,S,A)} (\mu_{A[a]} \bullet S) \equiv \sigma^{x(R,S,A)} \bullet \mu_{A[a]} \bullet (R \blacktriangleright\!\bowtie^{y(R,S)} S)$.

10.3. $R \blacktriangleright\!\bowtie^{p(R,S)} (\nu_{a}^{A} \bullet S) \equiv \nu_{a,L(R)}^{A} \bullet (R \blacktriangleright\!\bowtie^{p(R,S)} S)$.

10.4. $R \blacktriangleright\!\bowtie^{p(R,S,A)} (\nu_{a}^{A} \bullet S) \equiv \sigma^{x(R,S,A)} \bullet \nu_{a,L(R)}^{A} \bullet (R \blacktriangleright\!\bowtie^{y(R,S)} S)$.

B.3. The D-join Reduction Rules

Rule 11 (d-join reduction)

11.1. $R \blacktriangleright\!\bowtie^{p} S \equiv R \bowtie^{p} S$, where $S \not\prec R$;

11.2. $R \blacktriangleright\!\bowtie^{p} A_a \equiv \sigma^{p} \bullet \mu_{A[a]} \bullet R$, where $A \in L(R)$.